1 Úvod

Motivací pro tuto diplomovou práci byla nedokonalost STL formátu, který je používán standardně pro přenos trojúhelníkových sítí modelů určených pro výrobu v průmyslovém odvětví označovaném jako *rapidní prototypování*. Nedokonalost STL formátu spočívá v tom, že nedokáže zachytit strukturu trojúhelníkové sítě a při dalším zpracování této sítě, může dojít k nesprávnému určení struktury. Z důvodu narůstající velikosti používaných trojúhelníkových sítí bylo potřeba se pokusit o nalezení takových přístupů, s jejichž pomocí by se podařilo efektivně vytvořit z STL souboru trojúhelníkovou síť, provést její kontrolu, zda odpovídá požadavkům *rapidního prototypování* a následně určit vhodný způsob odstranění nedostatků sítě. Jedním z hlavních cílů bylo nalezení metody, která by urychlila odstranění vícenásobných výskytů stejných vrcholů v STL souboru.

2 Rapidní prototypování, formát STL a trojúhelníkové sítě

V průmyslu se stále více prosazuje odvětví označované jako *rapidní prototypování,* zkr. *RP* (angl. *Rapid Prototyping*). Toto odvětví se zabývá rychlým vyráběním prototypů (dále označován jako *RP-prototyp*) nově navrhovaných výrobků. Návrhář nejprve provádí návrh výrobku v CAD systému na počítači, kde, podle možností tohoto systému, je schopen dělat na navrhovaném výrobku potřebné testy a výpočty a také má možnost vizuálně kontrolovat, zda návrh odpovídá jeho záměrům. Ve chvíli, kdy potřebuje provést nějaké fyzické testy na výrobku nebo by potřeboval vidět jeho skutečné tvary, je nucen si nechat prototyp vyrobit. Při klasickém postupu by byl prototyp vyroben stejným postupem jako by byl vyráběn vlastní výrobek, což by si vyžádalo:

- zajištění potřebného materiálu,
- nastavení a seřízení strojů,
- čekání na dokončení výroby prototypu až několik dní, či dokonce týdnů.

Odstranění těchto nedostatků nabízí právě *RP*.

Výroba *RP-prototypu* trvá jen několik hodin a není nutné provádět zásah do stávajících výrobních linek výrobce, neboť výroba *RP-prototypu* se provádí na speciálním stroji (dále ozn. *RP-stroje*). Proces výroby probíhá postupným kladením tenkých vrstev materiálu o tloušťce cca desetiny milimetru. Tvar *RP-prototypu* může být téměř libovolný a neovlivňuje výrobní postup. Výroba se připravuje výhradně na počítači v CAD systému, ze kterého se před výrobou nechá vytvořit datový soubor obsahující model. Datový soubor je předzpracován a jsou vytvořeny obrazy jednotlivých vrstev a ty se pak přímo posílají do *RP-stroje*, který z nich vyrobí celý *RP-prototyp*. Jako standardní souborový formát pro přenos dat modelu se v *RP* využívá formát *STL* – zkratka pochází ze slova *stereolitografie* (angl. *stereolithography*), což je jedna z metod *RP* pro výrobu *RP-prototypu*.

Je nutné podotknout, že v současné době se *RP* již nevyužívá pouze k výrobě *RP-prototypů*, ale také např. pro výrobu těžko představitelných objektů, optimalizaci výroby, i k výrobě běžných výrobků (tedy nejen prototypů) a dále ve zdravotnictví pro výrobu protéz končetin, prostorových modelů srdce, otisků zubů, modelů kostí, atd.

2.1 Trojúhelníkové sítě

Každý objekt, resp. jeho počítačový model, je třeba pro zpracování v počítači nějak reprezentovat. Každá reprezentace má své výhody a nevýhody a podle nich je třeba uvážit, která z možností je pro dané účely nejvhodnější. Jednou z takových reprezentací je trojúhelníková síť (angl. *triangle mesh*). Zde se objekt modeluje tak, že jeho povrch je aproximován (samozřejmě s určitou chybou) rovinnými ploškami, v tomto případě trojúhelníky. Výhodou využití trojúhelníků je, že jde o útvary, které jsou vždy rovinné, tzn. není třeba je dále rozdělovat při zobrazování. Navíc většina moderních grafických subsystémů podporuje zobrazení právě pomocí trojúhelníků a nabízí pro ně různá urychlení.

Trojúhelníková síť jakožto modelová reprezentace obsahuje informace o trojúhelnících, o jejich vrcholech, o normálových vektorech trojúhelníků, příp. i vrcholů, dále by měla obsahovat informace o struktuře modelu, tedy o vzájemných sousednostech trojúhelníků. Samozřejmě jsou možné i další informace podle potřeby.

2.2 Formát souboru STL

Jak již bylo řečeno dříve STL formát slouží k přenosu modelů objektů určených pro výrobu *RP* procesem. V souboru je uložena neuspořádaná množina trojúhelníků, která tvoří souvislý povrch modelu. STL formát existuje jak v ASCII verzi, tak v binární verzi [8], jejichž struktura je popsána v následujících odstavcích.

Formát ASCII STL

V tomto typu souboru jsou použita pro jednotlivé sekce klíčová slova, která musí být uvedena malými písmeny. Není nutné, aby řádky byly odsazeny jako je tomu u následujícího popisu, to je pro jen lepší orientaci.
Struktura ASCII STL souboru vypadá takto – tučně jsou uvedena (povinná) klíčová slova a kurzívou nepovinné položky:

```
solid libovolný_text_např._označení_modelované_součástky
...
facet normal normal_x normal_y normal_z
 outer loop
  vertex vertex_1_x vertex_1_y vertex_1_z
  vertex vertex_2_x vertex_2_y vertex_2_z
  vertex vertex_3_x vertex_3_y vertex_3_z
 endloop
endfacet
...
endsolid libovolný_text_např._označení_modelované_součástky
```

kde normal_x, normal_y, normal_z jsou složky normálového vektoru trojúhelníku,
vertex_1_i, vertex_2_i, vertex_3_i jsou souřadnice jeho vrcholů.
Všechny tyto hodnoty jsou čísla s pohyblivou řádovou čárkou.

Formát binárního STL

Zde je soubor tvořen dvěma typy bloků. První blok je 84 bytová hlavička s touto strukturou:

Počet bytů	Význam položky
80	libovolný text
4	znaménkové celé číslo udávající počet trojúhelníků v souboru

Datový typ použitý pro udání počtu trojúhelníků v souboru omezuje jejich počet na 2,147,483,647. Za tímto blokem, který je v souboru pouze jednou, je uveden pro každý trojúhelník jeden blok obsahující:

Počet bytů	Význam položky
4	složka *x* normálového vektoru
4	složka *y* normálového vektoru
4	složka *z* normálového vektoru
4	souřadnice *x* prvního vrcholu
4	souřadnice *y* prvního vrcholu
4	souřadnice *z* prvního vrcholu
4	souřadnice *x* prvního vrcholu
4	souřadnice *y* prvního vrcholu
4	souřadnice *z* prvního vrcholu
4	souřadnice *x* prvního vrcholu
4	souřadnice *y* prvního vrcholu
4	souřadnice *z* prvního vrcholu
2	zarovnání bloku na 50 bytů

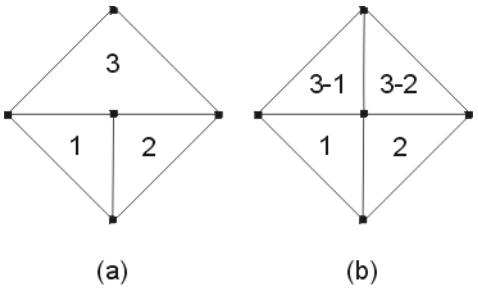
Všechny 4-bytové hodnoty v tomto bloku jsou čísla v pohyblivé řádové čárce (v jazyce typ C *float*). Pořadí bytů v jednotlivých položkách bloků odpovídá malému endianu (angl. *little endian*) – nejprve jsou uvedeny dolní dva byty a pak horní dva byty.

Pro STL soubor je předepsáno, že normálové vektory trojúhelníků směřují ven z povrchu a měli by být normalizovány. Vrcholy trojúhelníků by měli být uvedeny proti směru hodinových ručiček při pohledu z vnějšku povrchu. Směr normálových vektorů trojúhelníků odpovídá pravidlu pravé ruky.

Vzhledem k tomu, že pro jednotlivé trojúhelníky jsou v souboru uloženy pouze hodnoty souřadnic všech tří vrcholů a hodnoty složek normálového vektoru, není v STL formátu uložena jakákoliv informace o struktuře modelu objektu. Pro kontrolu konzistence modelu před výrobou a pro její zobrazení jsou však tyto informace nutné.

Vlastnosti STL formátu:
- Neobsahuje informace o struktuře součástky
- Může obsahovat více souvislých povrchů
- Povrchy se mohou vzájemně protínat
- Redundance dat – každý vrchol je v souboru uveden tolikrát v kolika je trojúhelnících
- Existuje ASCII i binární verze STL formátu

Ve specifikaci STL formátu je, kromě struktury souboru, dále uvedeno, že každé dva trojúhelníky musí mít společné právě dva vrcholy a tedy právě jednu hranu [6]. Tato podmínka se označuje jako Vrchol k vrcholu (angl. *Vertex-to-vertex*) a její význam je ilustrován na obr. 2.1.



Obr. 2.1 Pravidlo Vrchol k vrcholu
(a) Trojúhelník 3 sousedí přes 1 hranu se dvěma trojúhelníky – porušení pravidla Vrchol k vrcholu
(b) Řešení: rozdělení trojúhelníku 3

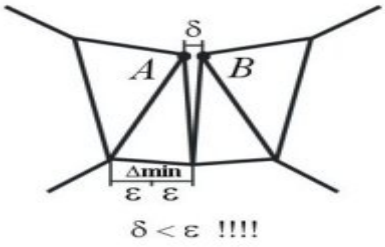
Je sice patrné, že formát STL je nedokonalý a že by bylo lepší využívat jiného formátu, který by obsahoval i informace o struktuře modelu (např. VRML, TRI), ale vzhledem k tomu, že je v *RP* využíván jako standardní formát, je nutné se jím zabývat. V dalším textu si uvedeme metody, kterými lze odstranit duplicity vrcholů v množině vrcholů, vytvořit ze STL souboru trojúhelníkovou síť včetně její kontroly a případné opravy. Dále si uvedeme metody pro výpočet normálových vektorů ve vrcholech trojúhelníkové síť, které nám umožní při jejím zobrazení využít i jiných metod stínování než jen konstantního (např. Gouraud).

3 Redukce duplicitních vrcholů v množině vrcholů

Jak již bylo řečeno, jednou z nevýhod STL formátu je, že každý vrchol je v něm uveden tolikrát, v kolika se nachází trojúhelnících. Je zcela evidentní, že tato redundance znamená zbytečné obsazení paměti, a proto bude naším cílem její odstranění. K odstranění násobných výskytů vrcholů potřebujeme při ukládání vrcholu kontrolovat, zda tento vrchol v množině vrcholů již existuje či nikoliv. Pokud zde ještě není, přidáme ho. Ale pokud zde již je, nepřidáváme ho znovu, a místo toho použijeme v množině již existující kopii.

3.1 Vzdálenost pro určení shodnosti vrcholů

Abychom mohli prohlásit dva vrcholy za shodné, musí oba ležet ve vzdálenosti menší než je prahová hodnota ε. Prahovou hodnotu ε můžeme určit například na základě vzorkovacího teorému, který říká, že vzorkovací frekvence musí být nejméně poloviční než je velikost nejmenšího požadovaného detailu. Pak bychom ε mohli určit jako poloviční délku nejmenší hrany v datech. Vrcholy, které k sobě budou blíže než ε, by pak mohly být prohlášeny za shodné.



Obr. 3.1 Nevhodné určení ε

Pokud by však nastala situace z obr. 3.1, kde dva vrcholy A a B, které nemají společnou hranu a jejich vzdálenost proto neovlivní hodnotu ε, jsou od sebe vzdáleny o vzdálenost δ, která je však menší než ε. V tom případě by při jejich porovnání byly prohlášeny za shodné a došlo by tak chybně ke ztrátě jednoho z nich.
Druhou možností, která nám pro určení hodnoty ε zbývá, je nastavit ji na nějakou vhodnou konstantní hodnotu, která by zajistila, že dva vrcholy budou prohlášeny za shodné pouze tehdy, půjde-li opravdu

o shodné vrcholy, jejichž souřadnice je „pouze“ zadána v různých trojúhelnících s nějakou odchylkou, např. 1,0 a 1,001. Pokud by v tomto případě bylo $\varepsilon = 0{,}002$, pak by oba vrcholy byly prohlášeny za shodné.

3.2 Metody prohledávání množiny vrcholů

Pro prohledávání množiny vrcholů při hledání duplicitních vrcholů máme tyto možnosti:
- Prohledání celé množiny vrcholů
- Řazení vrcholů podle hodnoty souřadnic
- Dělení obalového kvádru modelu
- Využití hashovací funkce

Prohledání celé množiny vrcholů

Jednou z možností zjišťování, zda vrchol již v množině je nebo není, je, pro každý vrchol na vstupu procházet celou množinu již uložených vrcholů a vrcholy postupně porovnávat s vkládaným vrcholem, dokud by nebyl nalezen shodný vrchol (s ohledem na prahovou hodnotu ε). Takový algoritmus by byl sice velice jednoduchý na implementaci, ovšem pokud by N byl počet různých vrcholů v datech a M byl počet všech vrcholů v původních datech (3 x počet trojúhelníků), pak by tento algoritmus měl složitost $O(M \cdot N)$. Z této složitosti je patrné, že algoritmus je více méně nepoužitelný již pro data obsahující řádově 10^4 trojúhelníků.

Řazení vrcholů podle hodnoty souřadnic

Abychom se vyhnuli neustálému prohledávání celé množiny vrcholů, musíme nějakým způsobem tuto množinu rozdělit nebo uspořádat a prohledávat vždy jen nějakou část. Například bychom mohli udržovat množinu vrcholů uspořádanou podle souřadnice X, pak podle Y a nakonec podle Z. Vhledem k tomu, že pro vyhledávání v uspořádané množině N prvků (což je počet různých vrcholů v trojúhelníkové síti) je složitost $O(\log(N))$, byla by celková složitost tohoto algoritmu $O(M \cdot \log(N))$, neboť by k vyhledávání došlo jednou pro každý vrchol z množiny všech (tedy včetně duplicitních) vrcholů, jejichž počet je M. Složitost je proto příznivější než u předchozího příkladu, ovšem nutnost udržovat množinu vrcholů uspořádanou s sebou přináší složitější implementaci.

Dělení obalového kvádru modelu

Naší další možností je určit obalový kvádr modelu (angl. *min-max box*), tedy kvádr o nejmenší velikosti, v němž jsou uzavřeny všechny vrcholy modelu. Tento kvádr následně rozdělit na krychličky a při zjišťování, zda vrchol již je nebo není v množině vrcholů, prohlížet pouze vrcholy spadající do stejné krychličky kvádru.

Při určování velikosti a tím i počtu krychliček musíme volit kompromis mezi touto velikostí a přijatelnými paměťovými nároky. Pokud bychom zvolili hranu krychličky takovou, že by byla menší (ale samozřejmě nenulová) než je nejmenší hrana v datech, pak by se v jedné krychličce (teoreticky) vyskytovaly pouze identické vrcholy. To by bylo velice výhodné, protože bychom zjistili, zda již vrchol je v množině vrcholů nebo není, jediným přístupem do obalového kvádru. Ovšem z hlediska paměťové náročnosti je tento způsob nepřípustný. Pokud by totiž velikost nejmenší hrany v modelu byla mnohonásobně menší než velikost nejmenší hrany obalového kvádru, např. $10^3\times$, pak by bylo potřeba alespoň 10^9 krychliček. Kdyby jednu krychličku představoval pouhý jeden byte, potřebovali bychom jen pro krychličky paměť o velikosti zhruba 1GB.

Hrana krychliček se proto volí v závislosti na datech, tak aby jejich počet byl odpovídající počtu vrcholů a zároveň paměťové nároky byly únosné. Obalový kvádr se rozdělí takto:

1. úprava obalového kvádru - zvětšení o ε (viz kap. 3.1)

$$\begin{aligned} x_{\min} &= x_{\min} - \varepsilon & x_{\max} &= x_{\min} + \varepsilon \\ y_{\min} &= y_{\min} - \varepsilon & y_{\max} &= y_{\max} + \varepsilon \\ z_{\min} &= z_{\min} - \varepsilon & z_{\max} &= z_{\max} + \varepsilon \end{aligned} \qquad (3.1)$$

2. výpočet velikosti *d* hrany krychličky, kde *N* je počet vrcholů v modelu

$$d = \sqrt[3]{\frac{(x_{\max} - x_{\min}) \cdot (y_{\max} - y_{\min}) \cdot (z_{\max} - z_{\min})}{N}} \qquad (3.2)$$

3. určení potřebného počtu k_x resp. k_y, k_z krychliček v ose v *x*, resp. *y*, *z* podle

$$k_x = \left\lfloor \frac{(x_{\max} - x_{\min})}{d} \right\rfloor + 1 \qquad (3.3)$$

Celkový počet *K* krychliček, který je zapotřebí pro redukci touto metodou, je dán jako

$$K = k_x \cdot k_y \cdot k_z \qquad (3.4)$$

To odpovídá situaci, kdy by vrcholy byly rozmístěny do pravidelné ekvidistantní mřížky. Tato situace ovšem ve skutečnosti ve většině případech nenastane. Naopak vrcholy budou někde více shluknuté jinde méně, některé krychličky mohou zůstat prázdné a jiné zase naopak budou obsahovat více různých vrcholů.

Nyní, poté co máme rozdělen obalový kvádr na krychličky, stačí určit ze souřadnic vrcholu, do které krychličky patří a poté ho porovnat (s ohledem na prahovou hodnotu ε) s vrcholy, které již máme v krychličce "uloženy". Tím dosáhneme složitosti *O(M·L)*, kde *L* je průměrný počet různých vrcholů připadající na jednu krychličku (*M* je opět počet všech vrcholů v původních datech) a mělo by platit $L << M$.

Nevýhodou této metody je, že v případě, kdy je jedna či více souřadnic vrcholu zadána v různých trojúhelnících s nějakou odchylkou, např. 1,0 a 1,001, mohou oba vrcholy „spadnout“ do různých krychliček. Následně by nedošlo k jejich porovnání a ačkoliv by obě verze, s ohledem na nastavené ε, mohli být považovány za jeden vrchol, budou v datech uloženy obě verze a navíc následně dojde v místě tohoto vrcholu k roztržení modelu (to je však řešeno dále v kap. 5).

Hashovací funkce

I u této metody je cílem vyhnout se procházení celé množiny vrcholů a hledání možných duplicitních vrcholů omezit pouze na nějakou podmnožinu vrcholů. Zde se k tomu využije hashovací funkce, která by na základě souřadnic vrcholu určila pozici v hashovací tabulce, na které by se hledali duplicitní vrcholy. V [2] byl navržen tento tvar hashovací funkce

$$\left(\left\lfloor 3 \cdot \left\lfloor |x| \cdot Q \right\rfloor / Q + 5 \cdot \left\lfloor |y| \cdot Q \right\rfloor / Q + 7 \cdot \left\lfloor |z| \cdot Q \right\rfloor / Q \right\rfloor \cdot T\right) \bmod T \qquad (3.5)$$

kde *x*, *y*, *z* jsou hodnoty souřadnic vrcholu,

Q určuje počet desetinných míst pro kvantování (v [2] nastaveno na 1000.0),

T je počet položek hashovací tabulky,

mod je operace vracející zbytek po celočíselném dělení.

Funkce tedy vypočte pozici v tabulce tak, že každou souřadnici nejprve zaokrouhlý dolů na 3 desetinná čísla (k tomu slouží parametr Q). Takto pozměněné souřadnice x', y', z' se vynásobí po řadě hodnotami 3, 5, 7 a pak se sečtou. Dále se to celé vynásobí velikostí tabulky, čímž se zajistí rozprostření vypočtených pozic po celé její délce. Nakonec se pozice upraví tak, aby nepřesahovala velikost tabulky pomocí operace *mod* vracející zbytek po celočíselném dělení. Pro každý vrchol pak procházíme pouze řetěz vrcholů patřící na stejnou pozici v tabulce a zjišťujeme, zda zde vrchol již existuje či nikoliv, obdobě jako jsme u dělení obalového kvádru prohledávali určenou krychličku.

Nyní ještě zbývá stanovit způsob pro výpočet potřebné velikosti T hashovací tabulky v závislosti na zpracovávaném modelu. Zde vyjdeme ze znalosti, že pro získání rozložení s očekávanými shluky délky 1.5 až 2.5 je třeba, aby počet prvků, které se do tabulky budou ukládat, odpovídal α = 30 až 50 % velikosti tabulky. Tuto hodnotu označujeme jako faktor naplnění tabulky (angl. *load factor*) [3]. Dále víme, že počet vrcholů N_V, které chceme zpracovat, je roven trojnásobku počtu trojúhelníků N_T. Není však třeba vytvářet tabulku o velikosti odpovídající počtu N_V, protože jsou zde započítány i duplicity vrcholů. Stačí nám tabulka, jejíž velikost bude odpovídat pouze různým vrcholům v datech, jejichž počet si označíme jako N_{DV}. N_{DV} určíme jako podíl N_V a průměrného počtu q_{avg} trojúhelníků sdílejících stejný vrchol. Je empiricky dokázáno, že q_{avg} je 6.

Výsledný vztah pro velikost tabulky T je

$$T = \frac{N_{DV}}{q_{avg}} \cdot \frac{1}{\alpha} \qquad (3.6)$$

a pro zvolené naplnění tabulky α = 0.5 dostaneme

$$T = \frac{N_{DV}}{6} \cdot \frac{1}{0.5} = \frac{3 \cdot N_T}{3} = N_T \qquad (3.7)$$

Velikost tabulky tedy přímo odpovídá počtu trojúhelníků v modelu.

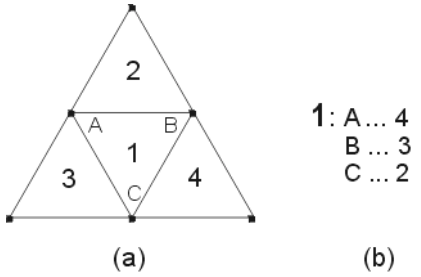
I u této metody dosáhneme složitosti $O(M \cdot L)$, kde L je průměrný počet různých vrcholů mapovaných na stejnou pozici tabulky, přičemž by opět mělo platit $L << M$. Také tato metoda má však nevýhodu, která byla uvedena u metody dělení obalového kvádru. Tou nevýhodou je, že pro jeden vrchol zadaný v různých trojúhelnících s odchylkou v souřadnicích může hashovací funkce určit rozdílný index do hashovací tabulky.

4 Vytváření informace o sousednostech trojúhelníků

Jednou ze základních informací obsažených v trojúhelníkové síti je informace o sousednostech trojúhelníků. Tato informace je potřebná pro rozpoznání struktury modelu, na jejímž základě se provádí mnoho operací nad trojúhelníkovou sítí (např. její zjednodušení, kontrola konzistence, výpočet normálových vektorů ve vrcholech, apod.) Bohužel, jak jsme si řekli již dříve, formát STL souboru tuto cennou informaci neobsahuje a vzhledem k tomu, že i my ji budeme v další práci potřebovat, uvedeme si nyní možnosti, kterými ji lze vytvořit z toho, co v STL souboru uloženo je. Nejprve si však řekneme něco o sousednosti trojúhelníků jako takové.

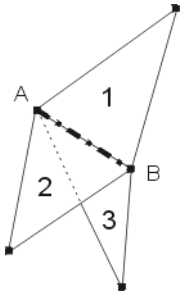
4.1 Sousednost trojúhelníků

Intuitivně samozřejmě význam této informace chápeme, ovšem je nutné si ji přesně definovat. Jako sousední označujeme takové dva trojúhelníky, které mají společnou hranu (viz obr. 4.1).



Obr. 4.1 Sousednost trojúhelníků
(a) Trojúhelník 1 má sousední trojúhelníky 2,3 a 4
(b) Označení sousedností pro trojúhelník 1

U každého trojúhelníku v trojúhelníkové síti máme uloženy odkazy udávající, se kterými trojúhelníky tento sousedí přes každou ze svých hran. Pro regulérní trojúhelníkovou síť musí platit, že každý trojúhelník má přes každou svou hranu nejvýše jeden sousední trojúhelník. Nesmí proto nastat situace, že by měly jednu hranu společnou více než dva trojúhelníky (viz obr. 4.2), což by znamenalo nejednoznačnost.



Obr. 4.2 Více než dva trojúhelníky sdílí hranu

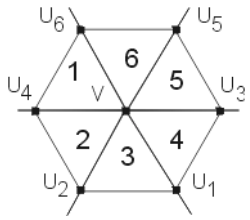
A pokud by se takový stav připustil, muselo by se jasně definovat, jakým způsobem se toto bude řešit. Pro účely *rapidního prototypování* je tento stav nepřípustný a my se proto budeme zabývat v dalších kapitolách tím, jak ho odstranit.

4.2 Metody vytváření informace o sousednostech trojúhelníků

Již tedy víme, že v STL souboru tato informace obsažena není a že je v něm pouze neuspořádaná množina trojúhelníků. Naštěstí se dá i na základě toho vytvořit informace o sousednostech. Máme k tomu dvě možnosti:
- Řazení pomocí košů
- Využití Voronoiových diagramů

Řazení pomocí košů

Tato metoda vychází z toho, že při redukci duplicity vrcholů v množině vrcholů se v trojúhelníkové síti ponechala pouze jedna kopie každého vrcholu a všechny trojúhelníky, které tento vrchol obsahují, se odkazují na tuto kopii. Pro každý vrchol V v trojúhelníkové síti tedy vytvoříme koš (angl. *bucket*) obsahující pouze trojúhelníky incidující s tímto vrcholem V (viz obr. 4.3).



Obr. 4.3 Incidence trojúhelníků s vrcholem V

Následně všechny trojúhelníky T_i v koši uspořádáme podle vrcholů U, které mají s vrcholem V společnou hranu, čímž dostaneme vedle sebe trojúhelníky obsahující stejnou hranu a tedy jsou sousední. Pokud se některý vrchol U_i v koši vyskytuje jen jednou, znamená to, že hrana VU_i je obsažena pouze v jednom trojúhelníku T_i, který tím pádem nemá sousední trojúhelník, a že zde proto dochází k porušení uzavřenosti roztržení trojúhelníkové sítě. Pro rapidní prototypování je neuzavřená trojúhelníková síť nepřípustná a my proto hranu VU_i budeme zpracovávat později při kontrole konzistence trojúhelníkové sítě (viz kap. 5).
Mohlo by se také stát, že by se v koši vyskytoval některý vrchol U_j více než dvakrát, což by naopak znamenalo, že hrana VU_j je obsažena ve více než dvou trojúhelnících. To je opět situace nepřípustná pro rapidní prototypování a my tuto hranu VU_j budeme opět zpracovávat při kontrole konzistence trojúhelníkové sítě.
Nyní si uvedeme jednotlivé části právě popsaného algoritmu.

Vytvoření košů pro jednotlivé vrcholy je jednoduché a daný algoritmus má složitost $O(N)$, kde N je počet trojúhelníků v trojúhelníkové síti:

Pro všechny trojúhelníky T_i
  Přidej T_i do košů pro všechny tři jeho vrcholy

Pro uspořádání trojúhelníků uvnitř košů podle vrcholů U_i lze použít některý algoritmus pro řazení, jejichž složitost je $O(M \cdot log(M))$, kde M je počet trojúhelníků uložených v daném koši. Vzhledem k tomu, že průměrný počet trojúhelníků incidujících s jedním vrcholem, což je průměrný počet trojúhelníků v jednom koši, je 6 (viz hodnota q_{avg} v kap. 3.2), lze pro řazení trojúhelníků uvnitř košů použít i algoritmus se složitostí $O(M^2)$, pokud to pro nás znamenalo např. snadnější implementaci.

A poslední část má díky uspořádání koše podle vrcholů U_i složitost $O(M)$. Vrchol U_j znamená, že jde o vrchol U ležící na j-té pozici v koši:

Pro každý vrchol U_j z koše
  Je-li $U_j \neq U_{j+1}$
    Přejdi na vrchol U_{j+1}

  Je-li $U_j \equiv U_{j+1}$ && je-li $U_j \neq U_{j+2}$
    Označ trojúhelníky obsahující hranu VU_j jako sousední
    Přejdi vrchol U_{j+2}

  Je-li $U_j \equiv U_{j+1}$ && je-li $U_j \equiv U_{j+2}$ && … && je-li $U_j \equiv U_{j+k-1}$ && je-li $U_j \neq U_{j+k}$
    Přejdi vrchol U_{j+k}

Využití Voronoiových diagramů

Další možností je využití *Voronoiových diagramů*. To by se ovšem vyplatilo tehdy, pokud by tato metoda byla použita již pro redukci duplicity vrcholů v množině vrcholů.
Při redukci duplicitních vrcholů by se *Voronoiův diagram* vytvářel postupným vkládáním vrcholů z jednotlivých trojúhelníků. Pokud by se zjistilo, že pro právě vkládaný vrchol V_1 v diagramu již existuje (s ohledem na prahovou hodnotu ε - viz kap. 3.1) shodný vrchol V_2, vrchol V_1 by se z množiny vrcholů

odstranil a byl by nahrazen vrcholem V_2. Pokud vrchol V_1 v diagramu ještě neexistoval, pak by se do něho přidal. Složitost algoritmu vytváření *Voronoiova diagramu* v třírozměrném prostoru je $O(N \cdot log(N) + N^2)$. Kromě toho, je tato metoda oproti metodě řazení pomocí košů, mnohem náročnější na implementaci.

5 Kontrola povrchů modelu

Pro výrobu prototypu některou z metod *rapidního prototypování* je důležité, aby jeho model byl složen pouze z uzavřených a vzájemně se neprotínajících povrchů. I přesto, že pro STL soubor je specifikováno, že má obsahovat pouze uzavřené povrchy, může se stát, že tomu tak z nějakého důvodu není. Může se také stát, že STL soubor obsahuje pouze uzavřené povrchy, ovšem při redukci vrcholů se nepodaří zredukovat všechny duplicity a některý z vrcholů zůstane zachován ve více kopiích. Pak v tomto vrcholu dojde k roztržení trojúhelníkové sítě, neboť algoritmus pro určení sousedností bude všechny jeho duplicitní kopie chápat jako odlišné vrcholy. Kromě toho by mohlo dojít k chybě při načítání STL souboru a tím např. ke ztrátě 1 či více trojúhelníků, což by opět mělo za důsledek neuzavřený povrch.

5.1 Metody pro odhalení chyb ve struktuře povrchů

Na odhalení možných chyb ve struktuře povrchů existuje několik jednoduchých metod [7]. Patří sem:

- Eulerovo pravidlo
- Pravidlo vrchol k vrcholu
- Kontrola počtu hran a trojúhelníků

Eulerovo pravidlo

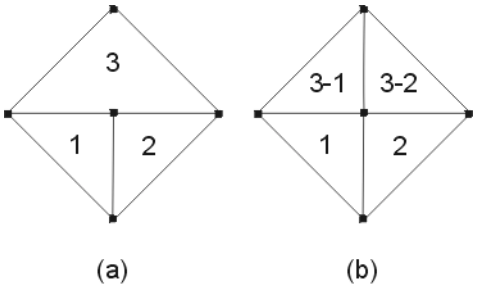
Základní metodou pro odhalení možné chyby ve struktuře je *Eulerovo pravidlo*, které má tvar

$$T + V - H = 2 \cdot P \qquad (5.1)$$

kde T je celkový počet trojúhelníků,
V je celkový počet vrcholů,
H je celkový počet hran,
P je celkový počet samostatných povrchů.

Pokud po dosazení do rovnice 5.1 vychází rovnost obou stran, pak je vše v pořádku a model obsahuje pouze uzavřené povrchy a je korektní pro použití v *rapidním prototypování*.

Pravidlo vrchol k vrcholu



Obr. 5.1 Pravidlo Vrchol k vrcholu
(a) Trojúhelník 3 sousedí přes 1 hranu se dvěma trojúhelníky – porušení pravidla Vrchol k vrcholu
(b) Řešení: rozdělení trojúhelníku 3

Toto pravidlo říká, že každá hrana v povrchu musí být sdílena právě dvěma trojúhelníky. Každý trojúhelník musí proto mít se svými sousedními trojúhelníky společné vždy právě dva vrcholy. Pokud nastane situace z obr. 4.1a, musí dojít k rozdělení trojúhelníku 3 tak, jak naznačuje obr. 4.1b.

Kontrola počtu hran a trojúhelníků

Na základě platnosti pravidla vrchol k vrcholu víme, že počet hran uzavřeného povrchu odpovídá 3/2 počtu trojúhelníků. Na základě toho získáme další tři podmínky pro bezchybnost struktury povrchu:

- počet trojúhelníků musí být sudý
- počet hran musí být násobek 3
- 2 * počet hran musí být roven 3 * počet trojúhelníků

Nebude-li některá z těchto podmínek splněna, znamená to, že model obsahuje 1 či více otevřených povrchů.

5.2 Určení počtu povrchů v modelu

Počet povrchů využijeme nejen při vyhodnocování pravidel pro odhalení chyb ve struktuře modelu, ale také při zjišťování průniku povrchů, při odstraňování děr v povrchu, apod. Abychom mohli povrchy spočítat, musíme nejprve zjistit, kterými trojúhelníky jsou jednotlivé povrchy tvořeny. Pro tyto účely využijeme znalosti sousedností trojúhelníků a na jejich základě projdeme vždy celý povrch a označíme si všechny jeho trojúhelníky. Algoritmus vypadá následovně:

Pro každý trojúhelník T_i
  Má-li již trojúhelník T_i nebo některý z jeho (existujících) sousedů S_1, S_2, S_3 určen identifikátor povrchu
    Urči nejmenší hodnotu h z těchto identifikátorů
    Přiřaď všem trojúhelníkům T_i, S_1, S_2 a S_3 jako identifikátor povrchu hodnotu h
  Nemá-li ani jeden z trojúhelníků T_i, S_1, S_2 a S_3 ještě přiřazen identifikátor povrchu
    Přiřaď všem trojúhelníkům T_i, S_1, S_2 a S_3 dosud nepřiřazený identifikátor povrchu
Opakuj celé znovu, pokud došlo u nějakého trojúhelníku ke změně identifikátoru jeho povrchu

Po provedení celého algoritmu, máme u všech trojúhelníků jednoho povrchu přiřazeny stejné identifikátory a podle počtu různých identifikátorů zjistíme, kolik je v modelu různých povrchů.

5.3 Možné typy povrchů

5.3.1 Uzavřené povrchy

Pro tyto povrchy nám u všech metod pro odhalení chyb ve struktuře povrchů vyjdou správné výsledky, což nás informuje o tom, že je všechno v pořádku a že každý trojúhelník má určeny všechny tři své sousedy. Strukturálně je všechno v pořádku, ovšem nyní je ještě nutné zjistit, zda je všechno v pořádku i z geometrického hlediska. Naším úkolem je určit, zda se některé povrchy navzájem neprotínají.

Nedochází-li k žádnému průniku povrchů, není již zapotřebí žádných dalších kontrol ani úprav, pouze by ze strany uživatele mohlo dojít k požadavku na transformace povrchů (otočení či posunutí) tak, aby při výrobě metodami *rapidního prototypování* nedocházelo k plýtvání materiálem.

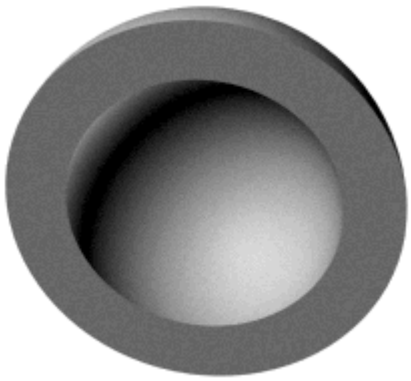
Pokud by však některé povrchy spolu mají geometrický průnik, je třeba učinit taková opatření, aby tento průnik byl odstraněn. Jestliže dochází ke geometrickému průniku povrchu A s povrchem B, pak máme následujících 5 možností, jak tento průnik odstranit:

- Vytvořit nový povrch C jako $A \cap B$
- Vytvořit nový povrch C jako $(A \cup B) - (A \cap B)$
- Vytvořit nový povrch C jako $A - B$
- Vytvořit nový povrch C jako $B - A$
- Posunutí jednoho či obou povrchů tak, aby platilo $A \cap B = \varnothing$

Kromě toho by u souvislých povrchů mohla nastat ještě situace, kdy je jeden povrch uzavřen uvnitř jiného (např. dvě soustředné koule s různými poloměry – viz obr. 5.2).

Obr. 5.2 Povrch uzavřený v jiném povrchu

Zde je nutné se zeptat uživatele, zda má povrch A skutečně být uvnitř povrchu B. Pokud ne, tak uživatel musí určit nějakou transformaci, která zajistí, že povrch A již nebude uvnitř povrchu B. Pokud uživatel s uzavřením povrchu A v povrchu B souhlasí, tak se jako výsledné těleso při *RP*-výrobě určí objem umístěný mezi povrchy A a B (viz obr. 5.3). U povrchu A bude pouze třeba změnit směr normálových vektorů trojúhelníků, aby směřovaly dovnitř ven ze vzniklého objemu.

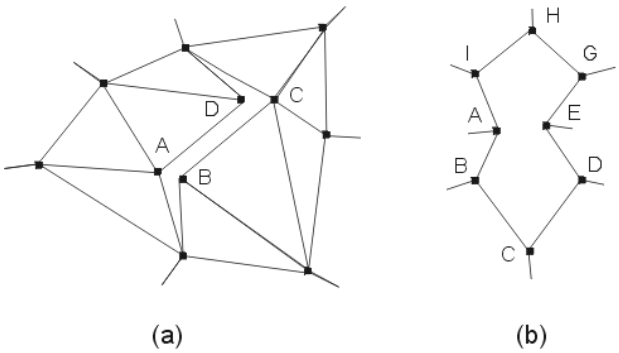


Obr. 5.3 Řez objektem vzniklým při uzavření jednoho povrchu v jiném

5.3.2 Neuzavřené povrchy

Neuzavřené povrchy jsou takové povrchy, které obsahují alespoň jeden trojúhelník, pro který nebyl nalezen sousední trojúhelník. Pro tyto povrchy nám u 1 či více metod pro odhalení chyb ve struktuře povrchů vyjde nesprávný výsledek a my tak víme, že je povrch neuzavřený, tedy že jsou v něm díry.

V povrchu mohou vzniknout díry dvojího typu. Jednak to jsou díry, které vznikly na základě nenalezení všech duplicitních kopií téhož vrcholu a došlo tak k roztržení trojúhelníkové sítě. Proto si tento typ děr označíme jako *neskutečné díry*. Druhý typ děr je naopak způsoben opravdovou nepřítomností některých trojúhelníků v povrchu a proto si tyto díry označíme jako *skutečné díry*. Mezi neskutečné díry zařadíme i takové díry, které nevznikly nedokonalou redukcí, ale jsou po jejich obvodu opravdu odlišné vrcholy, ovšem vzdálenost těchto vrcholů je menší než nějaká zvolená hodnota ε (nemusí se shodovat s prahovou hodnotou ε uvedenou v kap. 3.1) a lze je tak považovat za shodné. Příklady obou typů děr jsou vidět na obr. 5.4a,b.



Obr. 5.4 Typy děr v povrchu
(a) Neskutečná díra
(b) Skutečná díra

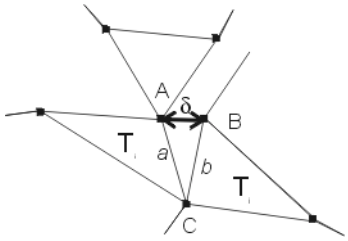
Mezi *neskutečné díry* lze zařadit také ty díry, které vzniknou na hranách nesplňujících pravidlo *vrchol k vrcholu* (viz obr. 5.1a). Jejich odstranění však nevyžaduje žádnou speciální metodu. Jak bylo uvedeno u definice pravidla *vrchol k vrcholu,* stačí trojúhelník 3 rozdělit na dva trojúhelníky podle obr. 5.1b.

Pro odstranění děr máme tři možnosti:
- Redukce vrcholů po obvodu díry
- Triangularizace díry
- Spojení více povrchů v jeden

Redukce vrcholů po obvodu díry

Zde je nutné po obvodu díry nalézt takovou dvojici vrcholů A a B, pro které platí (viz také obr. 5.4):

a) Vzdálenost $\delta = d(A, B) < \varepsilon$ (vrcholy musí ležet dostatečně blízko sebe)
b) $A \in$ trojúhelníku $T_i \wedge B \in$ trojúhelníku $T_j \wedge i \neq j$ (oba vrcholy patří do různých trojúhelníků)
c) Oba vrcholy *A*, *B* jsou koncovými vrcholy hran *a*, *b*, jejichž druhý koncový vrchol *C* je oběma hranám společný. Hrany *a*, *b* leží na obvodu díry.



Obr. 5.4 Redukce vrcholů po obvodu díry

Vrcholy *A* a *B* leží tedy dostatečně blízko, aby bylo možné prohlásit, že jsou shodné. Následně se trojúhelník T_j označí jako soused trojúhelníku T_i přes hranu *BC* a v trojúhelníku T_i se označí jako soused přes hranu *AC* trojúhelník T_j. Dále se v trojúhelníku T_j nahradí vrchol *B* vrcholem *A* anebo se v trojúhelníku T_i nahradí vrchol *A* vrcholem *B*. Po takovém odstranění vrcholu musíme hledat další dvojici vrcholů *A*, *B* pro odstranění. To celé se opakuje dokud se neprojde celý obvod díry, aniž by byla nalezena dvojice vrcholů *A*, *B*. Z požadavků na vrcholy *A* a *B* vyplývá, že tato možnost připadá v úvahu pouze u *neskutečných děr*.

Triangularizace díry

Tato možnost znamená vytvořit nad množinou vrcholů incidujících s dírou regulární triangulaci. Např. můžeme následující metodu:

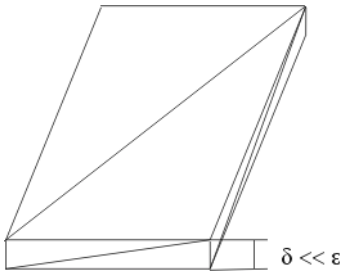
```
Existují-li po obvodu díry hrany AC a BC
  Je-li úhel svíraný těmito hranami konvexní
    Pokud hrana AB neprotíná nějakou část povrchu
      Do povrchu přidej trojúhelník ABC
      Z díry odstraň hrany AC a BC
      Nastav sousednosti přes hrany AC a BC
      Pokud neexistuje další trojúhelník obsahující hranu AB
        Hranu AB zařaď do obvodu díry
```

Spojení více povrchů v jeden

Tato možnost je určena pro případy, kdy jsou v STL souboru skutečně uloženy povrchy, které jsou na sebe vzájemně napojeny. Jako příklad takových povrchů si uveďme dva válce sdílející jednu podstavu . V tom

případě při určování sousedností (viz kap. 4) vzniknou tři samostatné povrchy – plášť válce *A*, plášť válce *B* a společná podstava. To však nelze chápat jako chybu algoritmu pro vytváření sousedností, ale jako důsledek nedokonalosti STL formátu, který v sobě neuchovává žádnou informaci o struktuře modelu. My nyní můžeme buď spojit povrchy obou plášťů a povrch podstavy z modelu odstranit anebo vytvořit kopii povrchu podstavy, kterou společně s původní podstavou propojíme s plášti válců. Která z možností se provede rozhodne uživatel, my pouze můžeme zjistit, které povrchy by bylo možné spojit.

5.3.3 Povrchy modelu s nulovým objemem, s nulovou plochou



Obr. 5.5 Model s nulovým objemem

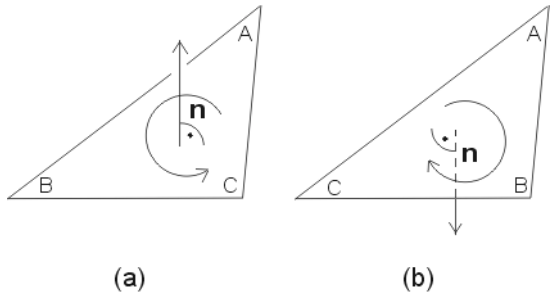
Může také nastat situace z obr. 5.5, kdy je povrch modelu zcela uzavřený a ačkoliv by měl takový model mít nenulový objem, není tomu tak z důvodu jeho nulové tloušťky, lépe řečeno menší než je nejmenší přípustná hodnota ξ. Tento případ nastane samozřejmě i u neuzavřených povrchů, které budou tvořeny pouze trojúhelníky ležícími v jedné rovině. Vzhledem k tomu, že jde opět o situaci nepřípustnou pro *rapidní prototypování*, je nutné povrch upravit tak, aby měl tloušťku *t*, která již bude přípustná.

U uzavřeného povrchu stačí pouze obě vrstvy oddělit a vzájemně je posunout o délku *t* ve směru normálového vektoru povrchu, čímž se vytvoří tloušťka *t*. Poté se musí ještě obě vrstvy povrchu opět spojit. To se provede jednoduchou triangularizací nad množinou vrcholů z okrajů povrchů *A* a *A*' tak, že propojíme každý vrchol z povrchu *A* s jeho protějškem z povrchu *A*'. Tím po obvodu objektu vytvoříme pravoúhelníky, které ještě rozdělíme úhlopříčkou buď z vrcholu *A* do vrcholu *B*' nebo z vrcholu *B* do vrcholu *A*'.
Pokud se jedná o případ s neuzavřeným povrchem, je řešení podobné. Zde si vytvoříme druhou vrstvu povrchu tak, že existující vrstvu zkopírujeme a opět ji posuneme o vzdálenost *t* ve směru normálového vektoru povrchu. Další postup je již totožný s případem pro uzavřený povrch. Tím získáme model s povrchem tvořeným regulární trojúhelníkovou sítí a s tloušťkou *t*, která splňuje požadavky *rapidního prototypování* na vyrobitelnost.

Samozřejmě se může stát, že povrch modelu bude mít plochu o velikosti menší než je nejmenší přípustná hodnota ξ. Takové povrchy se z modelu jednoduše vyloučí.

6 Orientace trojúhelníků v modelu

Orientací trojúhelníku se rozumí pořadí, v němž jsou uvedeny jeho tři vrcholy (viz obr. 6.1).



Obr. 6.1 Orientace trojúhelníků a odpovídající směr normálového vektoru
(a) orientace proti směru hodinových ručiček
(b) orientace po směru hodinových ručiček

Jak vidíme na obr. 6.1, ovlivňuje orientace trojúhelníku směr jeho vypočítaného normálového vektoru. U STL se přepokládá, že trojúhelníky mají normálové vektory směřující ven z povrchu. Nicméně není zaručeno, že tomu tak opravdu je a vzhledem k tomu, že pro výrobu metodou rapidního prototypování je důležité, aby normálové vektory trojúhelníků skutečně směřovaly ven z povrchu, je nutné zajistit správnou orientaci trojúhelníků tak, aby se normálové vektory vypočítaly správně. Pokud v STL souboru nejsou normálové vektory trojúhelníků uvedeny, pak se předpokládá, že se vypočítají podle pravidla pravé ruky, tedy že ven z povrchu budou směřovat normálové vektory těch trojúhelníků, které jsou orientovány proti směru hodinových ručiček (angl. *counterclockwise*). Problém je ovšem v tom, že orientace jednotlivých trojúhelníků mohou být různé a v jednom povrchu by se mohli vyskytnout i trojúhelníky s orientací po směru hodinových ručiček (angl. *clockwise*) a jejich normálový vektor by se pak vypočítal jako směřující dovnitř povrchu.

Abychom zajistili normálové vektory směřující ven z povrchu, musíme před jejich výpočtem nastavit u všech trojúhelníků povrchu stejnou orientaci. Tím budou všechny normálové vektory shodně směřovat buď ven z nebo dovnitř povrchu. Poté se na základě zobrazení modelu uživatel rozhodne, zda je tento směr správný nebo se mají změnit orientace povrchů a normálové vektory přepočítat.
Shodné orientace trojúhelníků dosáhneme pomocí následujícího algoritmu:

První trojúhelník každého povrchu označ jako orientovaný

Pro každý trojúhelník T_i
Je-li již trojúhelník T_i nebo některý z jeho (existujících) sousedů S_1, S_2, S_3 orientován
Nastav u dosud nezorientovaných trojúhelníků shodnou orientaci
Celý cyklus opakuj znovu, pokud při poslední iteraci došlo ke změně orientace některých trojúhelníků

Algoritmus je velice jednoduchý, ovšem jeho složitost je pro N udávající počet trojúhelníků v daném povrchu $O(N^2)$. Po provedení algoritmu mají všechny trojúhelníky každého povrchu takovou orientaci, jakou má jeho první trojúhelník.

7 Výpočet normálových vektorů ve vrcholech

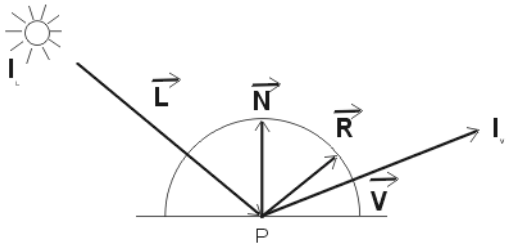
Ačkoliv se normálové vektory ve vrcholech trojúhelníkové sítě dají využít i k dalším účelům, v této diplomové práci budou vypočítávány pouze pro vylepšení vizuálního dojmu ze zobrazovaného modelu. U STL souboru je implicitně možné provést pouze konstantní stínování povrchu, neboť v něm jsou k dispozici

pouze normálové vektory trojúhelníků. Tzn. že pro celou plochu trojúhelníku bude vypočtena jedna hodnota intenzity světla na základě Phongova osvětlovacího modelu pro *M* světelných zdrojů [5]

$$I = I_A \cdot r_A + \sum_{k=1}^{M} I_{L_k} \cdot \left[r_s \cdot \left(\vec{V} \cdot \vec{R}_k \right)^h + r_d \cdot \left(\vec{L}_k \cdot \vec{N} \right) \right] \qquad (7.1)$$

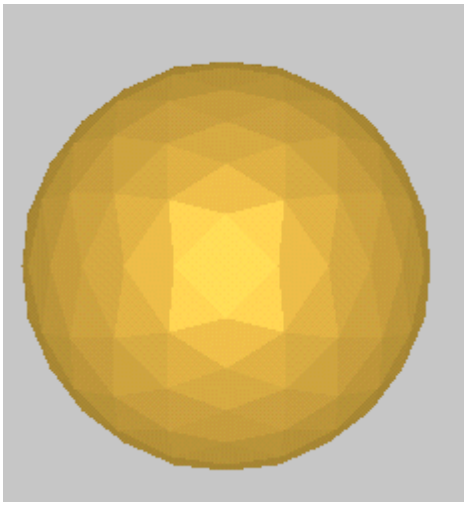
a tou bude vybarven vždy celý trojúhelník.
Jednotlivé hodnoty v rovnici 6.1 mají následující význam (viz také obr. 7.1):

I_A, I_{Lk} – intenzita okolního světla (angl. *ambient*), intenzita světla ze zdroje *k*
r_A, r_S, r_D – koeficienty odrazivosti povrchu trojúhelníku pro jednotlivé složky světla – okolní světlo (angl. *ambient*), zrcadlová složka (angl. *specular*), rozptylová složka (angl. *diffuse*)
h – koeficienty ostrosti zrcadlového odrazu
N – normálový vektor trojúhelníku
V – vektor určující směr pohledu pozorovatele
$\boldsymbol{N_k}$ – vektor určující směr paprsku z *k*-tého světelného zdroje po jeho odražení
$\boldsymbol{L_k}$ – vektor určující směr dopadu paprsku z *k*-tého světelného zdroje



Obr. 7.1 Odraz paprsku na povrchu tělesa (převzato z [5])

Jedna hodnota intenzity pro celý trojúhelník způsobí, že mezi dvěma trojúhelníky, které nejsou v jedné rovině, bude při zobrazení vidět ostrá hrana, ačkoliv v objektu modelovaném trojúhelníkovou sítí tato hrana ve skutečnosti vůbec není (viz obr 7.2).



Obr. 7.2 Konstantní stínování modelu koule

Řešením je použití dokonalejší metody stínování, jako je např. metoda Gouraudova či Phongova. Obě tyto metody vycházejí z normálových vektorů ve vrcholech trojúhelníků a na jejich základě vypočítávají intenzitu světla pro „každý“ bod povrchu trojúhelníku. Ačkoliv Phongova metoda dává lepší výsledky, protože provádí interpolaci normálových vektorů po ploše trojúhelníku a vždy vypočítává intenzitu světla podle rovnice 7.1 znovu, je naopak – právě díky tomuto – pomalejší. Naproti tomu Gouraudova metoda vypočítává intenzitu světla podle rovnice 7.1 pouze ve vrcholech trojúhelníku a intenzitu světla pro bod uvnitř plochy trojúhelníku určí jako trilineární interpolaci intenzit z vrcholů. Metoda je díky tomu rychlejší, ale nepodaří se jí zachytit odlesky uvnitř plochy trojúhelníku. Kromě toho Gouraudova metoda je rovněž značně podporována akcelerátory na grafických kartách.

Pozn.: V této diplomové práci se využívá pouze Gouraudova metoda stínování, neboť je použita v grafické knihovně OpenGL, která byla zvolena pro projekt VS 97155.

7.1 Výpočet normálových vektorů trojúhelníků

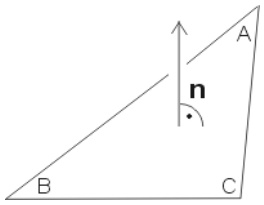
I přes to, že je tato kapitola určena pro výpočet normálových vektorů ve vrcholech (zkr. *NVV*), nejprve se zmíníme o výpočtu normálových vrcholů trojúhelníků (zkr. *NVT*) a to proto, že *NVV* budeme počítat na základě *NVT*.

V STL souboru by měly být *NVT* vypočítávány podle pravidla pravé ruky (viz kap. 6). Přesto, že pro samotný výpočet *NVT* není nutné respektovat okolní strukturu trojúhelníkové sítě, víme, že pro výrobu metodou *rapidního prototypování* je důležité, aby *NVT* směřovaly ven z povrchu. Musíme také zajistit konzistentní orientaci trojúhelníků sousedících s vrcholem z důvodu korektního výpočtu normálového vektoru v tomto vrcholu. Z toho plyne, že výpočet *NVT* by měl proběhnout teprve až po kontrole a nastavení správné orientace všech trojúhelníků v trojúhelníkové síti (viz kap. 6). Vlastní výpočet *NVT* proběhne podle následujícího algoritmu, jehož složitost je *O(N)*, viz obr. 7.3:

Pro každý trojúhelník T_i
  Je-li orientace trojúhelníku proti směru hodinových ručiček
    $\boldsymbol{u}$ = (A, B)
    $\boldsymbol{v}$ = (A, C)
  Je-li orientace trojúhelníku po směru hodinových ručiček
    $\boldsymbol{u}$ = (A, C)
    $\boldsymbol{v}$ = (A, B)

  $\boldsymbol{n} = \boldsymbol{u} \times \boldsymbol{v}$

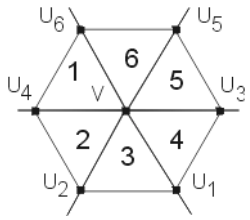
  Pokud je žádáno,
    normalizuj $\boldsymbol{n}$



Obr. 7.3 Normálový vektor trojúhelníku

7.2 Výpočet normálových vektorů ve vrcholech trojúhelníkové sítě

Metody používané k výpočtu *NVV* vycházejí ze znalosti *NVT* těch trojúhelníků, které s daným vrcholem incidují (viz obr. 7.4).



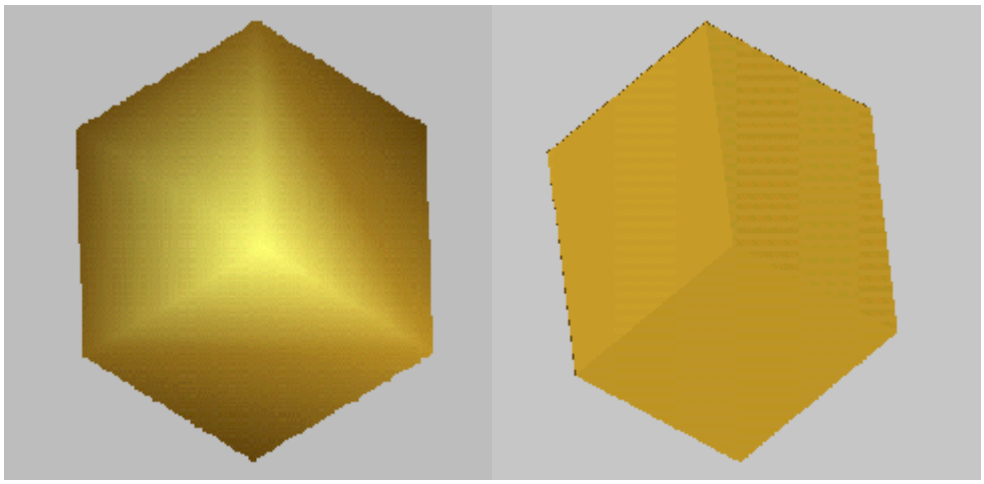
Obr. 7.4 Incidence trojúhelníků s vrcholem V

Tyto trojúhelníky tvoří tzv. trs (angl. *triangle fan*). U některých metod je nutné znát i uspořádání trojúhelníků v trsu podle sousedností, zatímco u jiných vystačíme s pouhou znalostí trojúhelníků, které se v trsu vyskytují. Na základě znalosti trsu trojúhelníků incidujících s daným vrcholem tedy lze pro výpočet *NVV* použít tyto 3 metody:

- Průměrný normálový vektor
- Vyhledání ostrých hran v trsu
- Proložení roviny koncovými vrcholy

Průměrný normálový vektor

Tato metoda jednoduše vezme pro vrchol *V* normálové vektory všech trojúhelníků, které s tímto vrcholem incidují a vypočte průměrnou hodnotu pro každou složku normálového vektoru. Metoda je tedy velice jednoduchá a je určena pro případy, kdy jednotlivé trojúhelníky v trsu leží téměř v jedné rovině. Jak je vidět na obr. 7.5 u modelu krychle dojde při takto počítaných *NVV* k vizuálnímu dojmu zakulacení v místě vrcholů. Lepších výsledků by se dalo dosáhnout, kdybychom v okolí vrcholů trojúhelníkovou síť zjemnili. To však s sebou přináší zvýšení nároků na paměť.



(a) (b)

Obr. 7.5 Krychle zobrazená se stínováním metodou Gouraud

(a) Metoda průměrného normálového vektoru
(b) Metoda Vyhledání ostrých hran

Vyhledání ostrých hran v trsu

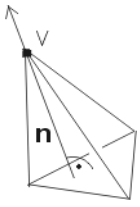
Tato metoda zohledňuje situace, kdy jsou úhly mezi sousedními trojúhelníky v trsu příliš ostré a *NVV* není vhodné vypočítávat jako průměr z normálových vektorů všech trojúhelníků v trsu. Místo toho se zde prochází celý trs, dokud se nenajde trojúhelník, který je již příliš odkloněn od prvního trojúhelníku v této části trsu (více než o úhel α). Mezi takto nalezenými částmi trsu se jakoby vytvoří ostrá hrana a vrchol pak má pro každou část trsu mezi dvěma ostrými hranami jinou hodnotu normálového vektoru, která se opět určí jako průměr – ovšem pouze z normálových vektorů trojúhelníků patřících do dané části trsu. Nevýhodu této metody je, že potřebuje pro každý vrchol paměť pro více než jeden normálový vektor, avšak výsledky získané touto metodou jsou lepší než u metody předchozí.

Pro všechny vrcholy
  $T_1 = T_i$ = libovolný trojúhelník z i-té části trsu
⎡ Dokud existuje (T_j = soused T_i) && $T_j \neq T_1$
⎢ Dokud existuje T_j && není úhel mezi NV trojúhelníků T_i a $T_j \geq \alpha$ && $T_j \neq T_1$
⎢ T_j = sousední trojúhelník
⎢
část A
⎢
⎢ Vypočti složky NV ***n*** jako průměry složek NV trojúhelníků T_i až T_{j-1}
⎢ Ulož NV ***n*** NV trojúhelníků T_i až T_{j-1}
⎢
⎣ $T_i = T_j$
 Pokud trojúhelník T_1 je sousedem T_j
  NVV ve vrcholu V v trojúhelníku T_j = NV T_j
  Přejdi na další vrchol

 Pokud T_j již souseda nemá (trs není uzavřen)
  Zopakuj část A pro opačnou část trsu od T_1

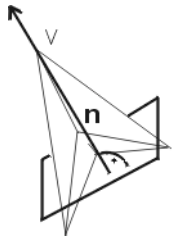
Proložení roviny koncovými vrcholy

Jak je uvedeno v [1] lze pro situaci, kdy je v trsu vrchol *V* spojen hranami se 3 vrcholy (viz obr. 7.6), využít tuto metodu, která jako *NVV* určí normálový vektor roviny procházející právě těmito vrcholy.



Obr. 7.6 Normálový vektor roviny proložené 3 vrcholy (převzato z [1])

Pokud by různých vrcholů bylo více než 3, pak by již nemuselo být možné proložit jimi rovinu, a proto se doporučuje využít metody nejmenších čtverců, s jejíž pomocí se proložená rovina určí (viz obr. 7.7).



Obr. 7.7 Normálový vektor roviny proložené více vrcholy (převzato z [1])

8 Úvod

V rámci této diplomové práce byla vytvořena samostatně spustitelná aplikace *STLViewer*. Dále byla vytvořena DLL knihovna s moduly pro MVE (*Modular Visualisation Environment*), což je aplikace, která vznikla v rámci projektu VS 97 155 a je určena pro vizualizace a zpracování obrazových dat, trojúhelníkových sítí a objemových dat.

Obojí bylo vytvářeno v prostředí *Microsoft Visual C++* s využitím principů objektově orientovaného programování a cílovou platformou je *Windows NT*. Pro zobrazování trojúhelníkových sítí byla využívána knihovna *OpenGL*, která byla zvolena pro projekt VS 97 155, pod nímž byla tato práce vytvářena.

Jak DLL knihovna tak aplikace *STLViewer* umožňují tyto operace:
- načtení množiny trojúhelníků z STL souboru
- redukce duplicitních vrcholů
- vytvoření informace o sousednostech
- určení povrchů
- nastavení orientace trojúhelníků povrchu
- výpočet normálového vektoru ve vrcholech

Aplikace *STLViewer* navíc umožňuje zobrazení trojúhelníkové sítě

V následujících kapitolách si popíšeme datové struktury, způsob implementace a dosažené výsledky u jednotlivých použitých metod.

9 Použité datové struktury

Datové struktury použité v modulech určených pro MVE jsou dány datovými strukturami pro trojúhelníkové sítě navržené pro projekt VS 97 155. V aplikaci *STLViewer* byly tyto struktury použity zčásti také, nicméně některé části byly vypuštěny a to proto, že neměli využití v aplikaci, která funguje samostatně. Nyní si uvedeme datové struktury použité v modulech určených pro MVE a poté si řekneme, čím se liší datové struktury použité v aplikaci *STLViewer*.

9.1 Datové struktury použité v modulech pro MVE

Datové struktury pro trojúhelníkovou síť modelu získaného ze vstupu modulu v prostředí MVE obsahují dva druhy položek – statické, které mají přidělen po celou dobu „života“ modulu pevný paměťový prostor, a dynamické položky, jejichž paměťový prostor se v průběhu „života“ modulu mění. Ve statických položkách jsou uloženy informace týkající se celé trojúhelníkové sítě modelu a jsou to:

- nejmenší a největší hodnoty souřadnic v celém modelu (obalový kvádr modelu)
- korekce pozice modelu – vektor složený z hodnot x, y a z, o které je třeba model posunout ve směru jednotlivých souřadných os pro získaní jeho požadované pozice
- korekce natočení modelu – vektor složený z úhlů α_x, α_y a α_z, o které je třeba model natočit podle jednotlivých souřadných os pro získaní jeho požadované polohy
- zkreslení (zkroucení) modelu – vektor složený z hodnot x, y a z, kterými je třeba vynásobit měřítko jednotlivých souřadných os pro získaní požadovaného tvaru modelu
- počet položek alokovaných v polích, které se vztahují k vrcholům trojúhelníkové sítě
- počet položek alokovaných v polích, které se vztahují k trojúhelníkům trojúhelníkové sítě
- počet platných položek uložených v polích, které se vztahují k vrcholům trojúhelníkové sítě
- počet platných položek uložených v polích, které se vztahují k trojúhelníkům trojúhelníkové sítě
- příznak platnosti trojúhelníkové sítě

Dynamické části struktur jsou alokovány vždy podle potřeby dané trojúhelníkové sítě, aby buď nedocházelo ke zbytečnému plýtvání pamětí anebo aby naopak nebylo třeba uložit více dat než by umožňovali pevně alokované struktury. Všechny tyto položky jsou vícesložkové (kromě stavových vektorů) a jsou uloženy tak, že každá složka má své samostatné pole. Mezi tyto položky patří:

- pole souřadnic vrcholů trojúhelníkové sítě – v současné době se používají čtyři souřadnice (X, Y, Z a 4. souřadnice)[1]
- pole složek *x*, *y* a *z* normálových vektorů ve vrcholech – zde má každý vrchol trojúhelníkové sítě jeden normálový vektor
- pole indexů na první trojúhelník, v němž je vrchol v datech obsažen
- vektor stavových slov vrcholů trojúhelníkové sítě – zde je uložen příznak, zda je daný vrchol platný či neplatný
- pole indexů na (tři) vrcholy každého trojúhelníku
- pole indexů na (tři) sousední trojúhelníky každého trojúhelníku
- pole složek *x*, *y* a *z* normálových vektorů v trojúhelnících
- pole s identifikátory, které určují povrch příslušející danému trojúhelníku
- vektor stavových slov trojúhelníků trojúhelníkové sítě – zde je uložen příznak, zda je daný trojúhelník platný či neplatný

Do dynamických datových struktur ještě patří seznam povrchů v modelu, ten je však využíván pouze uvnitř modulu, neboť MVE s povrchy v trojúhelníkových sítích nepracuje.

Pro položky udávající nějakou souřadnici (např. souřadnici vrcholu či složku normálového vektoru) byl zvolen datový typ jazyka C *double*, tedy 8 bytový typ pro čísla s pohyblivou řádovou čárkou pro zajištění dostatečné přesnosti (15 platných číslic) pro všechny možné moduly MVE.
Pro položky, které udávají nějaký počet nebo index (např. počet vrcholů trojúhelníkové sítě, index na vrchol trojúhelníku, apod.) byl zvolen datový typ jazyka C *long*, tedy 4 bytový znaménkový typ pro celá čísla, což umožňuje uložení celého čísla nejvýše o hodnotě 2,147,483,647. Tento rozsah je důležitý, protože se očekává využívání MVE a jeho modulů pro trojúhelníkové sítě obsahující řádově 10^6 trojúhelníků a více.

9.2 Datové struktury použité v aplikaci *STLViewer*

Datové struktury použité v aplikaci *STLViewer* vycházejí ze struktur navržených pro moduly MVE, ovšem je zde několik rozdílů. Co se týče datových typů byl pro položky udávající nějakou souřadnici (např. souřadnici vrcholu či složku normálového vektoru) zvolen datový typ jazyka C *float*, tedy 4 bytový typ pro čísla s pohyblivou řádovou čárkou s přesností 7 platných číslic. Tento typ byl zvolen, protože aplikace bude zpracovávat STL soubory, pro které je předepsána přesnost všech souřadnic právě v rozsahu tohoto typu.
Pro položky, které udávají nějaký počet nebo index (např. počet vrcholů trojúhelníkové sítě, index na vrchol trojúhelníku, apod.) byl zvolen datový typ jazyka C *unsigned long*, tedy 4 bytový neznaménkový typ pro celá čísla, což umožňuje uložení celého čísla nejvýše o hodnotě 4,294,967,295. Tento typ byl zvolen pro zajištění maximálního rozsahu (ze stejných důvodů jako u modulů pro MVE – počet trojúhelníků řádově 10^6 a více) a vzhledem k tomu, že se nepředpokládá využití záporných hodnot.
Oproti strukturám pro moduly MVE zde byl přidán do dynamické části dat vektor indexů na normálový vektor pro každý vrchol trojúhelníku. Tedy místo toho, aby každému vrcholu odpovídal normálový vektor ležící v poli na stejné pozici jako daný vrchol, tak se ke každému trojúhelníku uloží odkaz na normálový vektor každého jeho vrcholu. Tato změna je zde proto, že v rámci diplomové práce byl implementován i algoritmus pro výpočet normálových vektorů ve vrcholech trojúhelníkové sítě. A protože je elementem trojúhelníkové sítě trojúhelník, je nutné pro jeho stínované zobrazení metodou *Gouraud* (viz kap. 7), aby byla k dispozici hodnota normálového vektoru ve všech třech vrcholech. Pokud by však měl každý vrchol pouze jeden normálový vektor, bylo by možné jej vypočítat pouze jako průměr z normálových vektorů s vrcholem

[1] Pro 4. souřadnici není v této diplomové práci opodstatnění, a proto není její pole využíváno. Z důvodů kompatibility s ostatními moduly MVE je však toto pole zařazeno v datových strukturách a nastaven příznak, že tato souřadnice neexistuje (zde je stačí nastavení ukazatele na toto pole na hodnotu NULL)

incidujících trojúhelníků. Pro implementaci i jiné metody je proto nutné, aby každý vrchol mohl mít pro každý trojúhelník jiný normálový vektor.

10 Redukce duplicitních vrcholů v množině vrcholů

10.1 Vzdálenost pro určení shodnosti vrcholů

Jak bylo zmíněno v teoretické části (viz kap. 3.1), jednou z možností pro určení prahové vzdálenosti ε, je tak učinit na základě délky nejmenší hrany v modelu. Pro určení nejkratší hrany je možné použít např. *eukleidovskou vzdálenost*, kdy pro vzdálenost *d* bodů A a B platí

$$d = d(A,B) = \sqrt{(B_x - A_x)^2 + (B_y - A_y)^2 + (B_z - A_z)^2} \qquad (10.1)$$

Vzhledem k tomu, že operace mocnina a odmocnina z čísel s pohyblivou řádovou čárkou patří mezi výpočetně náročnější a my se snažíme, aby redukce duplicit vrcholů byla, co nejrychlejší, lze místo eukleidovské vzdálenosti užít *vzdálenost šachovnicovou*. Zde pro vzdálenost *d* bodů A a B platí

$$d = d(A,B) = \max\left\{|B_x - A_x|, |B_y - A_y|, |B_z - A_z|\right\} \qquad (10.2)$$

Podle porovnání průměrných dob výpočtů délek 300.000 hran (viz tab. 10.1) je vidět, že výpočet délky hrany při použití šachovnicové vzdálenosti je oproti použití eukleidovské přibližně dvakrát rychlejší.

eukleidovská vzdálenost	0.00425 s
šachovnicová vzdálenost	0.00764 s

tab. 10.1 Porovnání průměrných časů vyhodnocení délky 300.000 hran (což odpovídá modelu o 100.000 trojúhelnících)

Naproti tomu šachovnicová vzdálenost „znevýhodňuje“ body tvořící vektor, jenž není rovnoběžný ani s jednou ze souřadných os. V takovém případě vyjde vzdálenost bodů menší než ve skutečnosti je.
Oba přístupy však dávaly stejné výsledky, tzn. že počet vrcholů po redukci duplicitních byl vždy stejný. Z porovnaných hodnot je patrné, že obě metody nabízejí stejné výsledky a že do počtu trojúhelníků řádu menšího 10^8 jsou časové rozdíly zanedbatelné. Od řádů vyšších je výhodnější použít výpočet délky hrany pomocí šachovnicové vzdálenosti.

Při použití prahové hodnoty ε jako poloviny nejkratší hrany v modelu (bez ohledu na metodu výpočtu délky hrany) docházelo ke ztrátám vrcholů, tedy že některé rozdílné vrcholy byly určeny jako shodné a došlo k odstranění jednoho z nich z množiny vrcholů. Proto bylo také použito nastavení prahové hodnoty ε na konstantní hodnotu 0,000001, při které ke ztrátě vrcholů nedošlo.

10.2 Metody prohledávání množiny vrcholů

Z algoritmů, které byly zmíněny v teoretické části, byly pro prohledávání množiny vrcholů při redukci duplicity vrcholů vybrány metody využívající dělení obalového kvádru modelu a využívající hashovací funkce. Obě metody byly vybrány pro jednoduchost implementace a také z důvodu příznivé složitosti ($O(M \cdot L)$, více viz kap. 3). Přesto, že mají obě metody shodnou složitost a mohlo by být vhodnější pro srovnání implementovat místo jedné z nich jinou metodu s jinou složitostí, byly implementovány právě tyto dvě metody, protože parametr *L* ve výrazu pro složitost je závislý u obou metod na něčem jiném. U metody dělení obalového kvádru modelu je *L* závislé na rozmístění vrcholů v prostoru a dá se tedy ovlivnit jedině

volbou hustoty mřížky obalového kvádru. Naproti tomu u metody využívající hashovací funkce je L závislé také na zvolené hashovací funkci a máme tak možnost výsledek ovlivnit. Cílem této části diplomové práce bylo zjištění vhodnosti využívání hashovací funkce pro tyto účely a příp. nalezení vhodného tvaru funkce, který by zajistil L nejlépe řádu 10.

Vzhledem k tomu, že u obou metod se na základě souřadnic vrcholů vypočte hodnota pozice v mapovacím vektoru, na které se posléze hledá duplicitní vrchol, popíšeme si nejprve implementaci mapovacího vektoru a výpočtu pozice jednotlivými metodami. Následně popíšeme implementaci redukce duplicitních vrcholů na základě získané pozice.

Dělení obalového kvádru modelu

Ve funkci implementující tuto metodu redukce se nejprve provede úprava hodnot souřadnic X_{min}, X_{max}, Y_{min}, Y_{max}, Z_{min}, Z_{max} obalového kvádru (samozřejmě jeho lokální kopie) podle rovnice 3.1 a dále se postupně podle rovnic 3.2, 3.3 a 3.4 vypočte velikost hrany d jedné krychličky, potřebný počet krychliček k_x, k_y a k_z v osách x, y a z a celkový počet krychliček, který je zapotřebí pro redukci touto metodou. I přesto, že obalový kvádr je třírozměrný, jsme schopni ho, s využitím vhodné mapovací funkce, nahradit vektorem a vyhneme se tak nutnosti implementovat ho jako třírozměrné pole.
V dalším kroku si tedy metoda alokuje vektor V o velikosti K položek, jejichž typ a význam bude popsán dále v kapitole *Implementace redukce duplicitních vrcholů.*
Použitá funkce mapující vrcholy do vektoru V má tvar

$$pozice = \lfloor (x - X_{\min}) \cdot d^{-1} \rfloor + \left(\lfloor (y - Y_{\min}) \cdot d^{-1} \rfloor + \lfloor (z - Z_{\min}) \cdot d^{-1} \rfloor \cdot k_y \right) \cdot k_x \qquad (10.3)$$

kde d^{-1} je převrácená hodnota velikosti hrany jedné krychličky, jejímž použitím (místo d) se vyhneme výpočetně náročné operaci dělení a můžeme použít rychlejší násobení.

Hashovací funkce

Ve funkci implementující tuto metodu redukce se nejprve alokuje hashovací tabulka v podobě vektoru V o velikosti T položek určené podle rovnice 3.7. Typ a význam položek mapovacího vektoru bude popsán dále v kapitole *Implementace redukce duplicitních vrcholů*.
Nyní si uvedeme tvary hashovacích funkcí použitých pro výpočet pozice v hashovací tabulce.

V teoretické části (viz kap. 3.2) byla uvedena hashovací funkce podle [2]. Při testování této funkce jsme zjistili, že neprovede rozprostření prvků po celé tabulce a vytváří poměrně velké shluky na některých pozicích tabulky, kde max. hodnota přesahovala 100 vrcholů na jednu pozici v tabulce. Vzhledem k tomu, že největší časovou zátěž tvoří při redukci právě prohledávání vrcholů na dané pozici tabulky, bylo třeba upravit hashovací funkci tak, aby byly vytvářeny nejvíce shluky délky 1 a aby se s narůstající délkou jejich počet (pokud možno rapidně) snižoval, přičemž by největší délka shluku byla řádu 10. Tím by bylo dosaženo dalšího zkrácení doby zpracování vrcholů celého modelu.

Změnou kvantování (parametr Q ve vzorci 3.5) jsme požadované změny nedosáhli a tak bylo třeba změnit celý tvar funkce. Z původní funkce bylo nejprve odstraněno kvantování hodnot souřadnic na určitý počet desetinných míst, protože tím se zbytečně odstranily rozdíly v souřadnicích vrcholů, které nám pomáhají generovat rozdílné pozice v tabulce.
Tím jsme dostali funkci tvaru

$$pozice = \lfloor (\alpha \cdot x + \beta \cdot y + \gamma \cdot z) \cdot C + 0.5 \rfloor \; AND \; T \qquad (10.4)$$

kde T je velikost tabulky,
C je konstanta (viz dále),
α, β a γ jsou koeficienty hashovací funkce, jejichž hodnoty jsme ponechali 3, 5 a 7,
AND je operace logického součinu.

Konstanta C má zajistit „roztažení“ hodnot vypočtených pozic na rozsah 4-bytového neznaménkového čísla tedy <0, $2^{32} - 1$>. Protože maximální hodnota ξ, kterou budeme konstantou C násobit je

$$\xi = \alpha \cdot X_{\max} + \beta \cdot Y_{\max} + \gamma \cdot Z_{\max} \qquad (10.5)$$

Poté, aby nedošlo k přetečení rozsahu 32 bitů, musí pro konstantu C platit, že

$$C_1 \cdot \xi \leq 2^{32} - 1 \qquad (10.6)$$

a tedy

$$C_1 \leq \frac{2^{32} - 1}{\xi} \qquad (10.7)$$

Dále pro konstantu C platit že její hodnota musí být taková, aby z vypočtené hodnoty ($\alpha \cdot X + \beta \cdot Y + \gamma \cdot Z$) byl zachován nejméně takový počet bitů, který odpovídá velikosti tabulky T. A proto

$$C_2 = 2^{32} - 2^k \qquad (10.8)$$

Spojíme-li nyní obě podmínky, pak pro C platí

Před oříznutím desetinné části, což je realizováno pomocí přetypování na datový typ jazyka C *unsigned long*, ještě přičteme konstantu 0.5, čímž zajistíme, že část vrcholů, pro něž vycházela desetinná část v rozsahu <0.5, 1.0), se přesunou o 1 pozici dále a zvýšíme tak pravděpodobnost pro rovnoměrné rozložení po celé tabulce.
Abychom mohli nahradit původně použitou operaci *mod*, která má zajistit, že vypočtená pozice nepřesáhne rozsah tabulky T, operací logického součinu *AND*, je nutné velikost hashovací tabulky, vypočtené podle rovnice 3.7, zaokrouhlit na hodnotu nejbližší vyšší mocniny 2.

Původní hashovací funkce byla vytvořena (i po našich úpravách) ze 4 přetypování, 6 násobení, 2 součtů a 1 logického součinu. Námi navržená hashovací funkce je tvořena z 1 přetypování, 3 násobení, 3 součtů a 1 logického součinu a lze s její pomocí dosáhnout shluků s max. hodnotou v řádu 10 (viz tab. 9.3a, b, c)[2].

Implementace redukce duplicitních vrcholů funkce

V této kapitole je popsána implementace algoritmu pro redukci duplicitních vrcholů v datech trojúhelníkové sítě. Konkrétně je zde popsána ta část, která je pro metody redukce dělením obalového kvádru a pomocí hashovací funkce shodná.

Nejprve se provede alokace mapovacího vektoru (hashovací tabulky nebo krychliček obalového kvádru), jak bylo popsáno na začátku předchozích dvou kapitol. Dále jsme se v předchozích kapitolách zmínili o položkách mapovacího vektoru, které si nyní popíšeme.

V obou případech se jedná o strukturu obsahující dvě položky:
- počet (neduplicitních) vrcholů uložených na dané pozici mapovacího vektoru
- index do pole vrcholů na další vrchol uložený na dané pozici mapovacího vektoru

Položka s počtem vrcholů je zde spíše z důvodu výpočtu statistik a pokud by tyto statistiky nebyly potřebné, mohla by se tato položka vypustit. To, že na dané pozici mapovacího vektoru dosud žádný vrchol neleží, by

[2] Výsledky této části diplomové práce, týkající se námi navržené hashovací funkce, budou prezentovány na [9]

se pak poznalo na základě nějaké speciální hodnoty indexu na další vrchol uložený na dané pozici vektoru (např. -1).

Vzhledem k tomu, že do položky mapovacího vektoru lze uložit index pouze na jeden vrchol, alokuje se ještě jeden vektor. Tento pomocný vektor je složen ze stejných položek jako samotný mapovací vektor a jejich počet odpovídá počtu vrcholů před redukcí. Ve chvíli, kdy se pro nově vkládaný vrchol A zjistí, že na vypočtené pozici mapovacího vektoru je již uložen odkaz na nějaký vrchol B a že tento vrchol B není s vrcholem A shodný, pak se právě do pomocného vektoru uloží na pozici patřící vrcholu B odkaz na vrchol A. Při vkládání dalšího vrcholu C na stejnou pozici by se odkaz na C ukládal do pomocného vektoru na pozici vrcholu A. Tím si zajistíme dostupnost všech vrcholů patřících na stejnou pozici mapovacího vektoru. V následujícím popisu metody redukce duplicitních vrcholů budeme pro jednoduchost označovat všechny vrcholy, které se mapují na pozici P mapovacího vektoru, jako vrcholy uložené na pozici P, ačkoliv ve skutečnosti je zde uložen pouze jeden.

Při redukci vrcholů se udržuje pozice posledního „neduplicitního“ vrcholu a každý další nový vrchol se přesune za tuto pozici. Duplicitní vrcholy tuto pozici neovlivní a tak po proběhnutí redukce nad všemi vrcholy, vznikne v poli vrcholů kompaktní množina různých vrcholů. V dalším popisu bude tato operace označena jako zařazení vrcholu za poslední „neduplicitní“ vrchol.

Nyní si popíšeme celou metodu redukce duplicitních vrcholů. Pro mapovací vektor budeme používat zkratku *MV*.

```
Pro všechny trojúhelníky Ti
  Pro každý vrchol Ai aktuálního trojúhelníku Ti
    Zjisti souřadnice vrcholu a vypočti pozici P v MV
    Jestliže ještě na pozici P v MV není uložen žádný vrchol
      Vlož do MV na pozici P vrchol Ai
    Jestliže již je nějaký vrchol na pozici P v MV uložen
      Ve smyčce projdi všechny vrcholy Bi uložené v MV na pozici P
        Je-li vzdálenost vrcholu Ai a vrcholu Bj menší než ε
          Nahraď v trojúhelníku T vrchol Ai vrcholem Bj
          Ukonči smyčku
       Pokud nebyl ve smyčce nalezen žádný vrchol Bi shodný s vrcholem Ai
         Přidej do MV na pozici P vrchol Ai

    Pokud nebyl nalezen žádný vrchol Bi shodný s vrcholem Ai anebo je Ai první vrchol na pozici P v MV
      Přičti nový vrchol k počtu vrcholů na pozici P v MV
      Zařaď vrchol Ai za poslední „neduplicitní“ vrchol a v trojúhelníku T aktualizuj odkaz na vrchol Ai
      Aktualizuj počet rozdílných vrcholů

uvolnění paměti použité pro pomocný mapovací vektor
uvolnění nadbytečných položek z pole vrcholů
počet vrcholů nastaven na počet různých vektorů po redukci duplicitních
nastavení příznaku, že vrcholy byly zredukovány
```

10.3 Dosažené výsledky

Nyní si ukážeme výsledky dosažené jednotlivými metodami. Budeme porovnávat výsledný faktor zaplnění mapovacího faktoru a velikosti shluku, což je počet různých vrcholů připadající na stejnou pozici mapovacího vektoru. Jako testovací modely posloužily tři STL soubory s následujícími charakteristikami:

	Počet trojúhelníků	Původní počet vrcholů	Počet vrcholů po redukci
CTHead.stl	555,411	1,666,233	278,856
Teapot.stl	159,600	478,800	80,202
Delaunay.stl	199,915	599,745	100,000

Tab. 10.2 Charakteristiky použitých modelů

Použitá technika	Velikost mapovacího vektoru	Koeficient zaplnění [%]	Nejmenší velikost shluku	Průměrná velikost shluku	Největší velikost shluku
Dělení obalového kvádru	1690368	6.446	1	2.5591	12
Původní hashovací funkce	1048576	0.096	1	277.1928	356
Nová hashovací funkce	1048576	23.248	1	1.1540	6

Tab. 10.3a Porovnání výsledků redukce duplicitních vrcholů v modelu CTHead.stl

Použitá technika	Velikost mapovacího vektoru	Koeficient zaplnění [%]	Nejmenší velikost shluku	Průměrná velikost shluku	Největší velikost shluku
Dělení obalového kvádru	486837	3.918	1	4.2043	127
Původní hashovací funkce	262144	0.383	1	79.8825	109
Nová hashovací funkce	262144	26.299	1	1.1634	5

Tab. 10.3b Porovnání výsledků redukce duplicitních vrcholů v modelu Teapot.stl

Použitá technika	Velikost mapovacího vektoru	Koeficient zaplnění [%]	Nejmenší velikost shluku	Průměrná velikost shluku	Největší velikost shluku
Dělení obalového kvádru	614125	1.331	3	5.9974	14
Původní hashovací funkce	1048576	0.385	5	99.2063	133
Nová hashovací funkce	1048576	31.693	1	1.2036	6

Tab. 10.3c Porovnání výsledků redukce duplicitních vrcholů v modelu Delaunay.stl

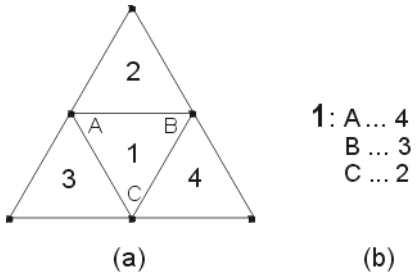
Z předložených tabulek vidíme, že námi navržený tvar hashovací funkce dává velmi dobré výsledky. Velikost shluku se stabilně drží v řádu jednotek. Také výsledný faktor naplnění je u této metody okolo 30 %, což podle [3] umožňuje průměrnou délku shluku 1.5 až 2.5.

11 Vytváření informace o sousednostech trojúhelníků

Jedním z cílů této diplomové práce bylo vytvoření informace o sousednostech v trojúhelníkové síti získané z STL souboru. V aplikaci *STLViewer* je tato informace dále využívána pro určení počtu samostatných povrchů (viz kap. 5.2) a při výpočtu normálových vektorů ve vrcholech modelu reprezentovaného trojúhelníkovou sítí (viz kap. 7). V rámci MVE může být tato informace využita dále např. v modulu, který provádí zjednodušení trojúhelníkové sítě.
Z metod zmíněných v teoretické části byla zvolena metoda řazení pomocí košů (viz kap. 3.2) z důvodu nenáročné implementace. Mimo to byl její výběr ovlivněn také tím, že tato metoda při své činnosti navazuje na redukci duplicitních vrcholů (viz dále) a protože se během testů ukázalo, že metoda pro redukci využívající hashovací funkce vykazuje velmi dobré výsledky, nebylo třeba hledat další alternativu v podobě využití metody *Voronoiových diagramů* (viz kap. 3.2) pro redukci duplicitních vrcholů a následně i pro vytvoření informace o sousednostech.

Způsob značení a uložení informace o sousednosti trojúhelníků

V každém trojúhelníku je informace o sousednostech tvořena třemi odkazy na trojúhelníky, které mají s tímto trojúhelníkem společnou hranu. Z důvodu jednoznačnosti je jako *i*-tý sousední trojúhelník trojúhelníku *T* označen trojúhelník mající s ním společnou tu hranu, která neobsahuje jeho *i* -tý vrchol (viz obr. 11.1).



Obr. 11.1 Sousednost trojúhelníků
Trojúhelník 1 má pro vrchol A sousední trojúhelník 4,
pro B trojúhelník 3 a pro C trojúhelník 2

Jak jsme si řekli již v kapitole 4.2 je nepřípustné, aby jednu hranu sdílely více než dva trojúhelníky. Dokonce pro *rapidní prototypování* je důležité, aby jednu hranu sdíleli právě dva trojúhelníky. Při vytváření informace o sousednostech se budeme řídit pouze prvním omezením, tedy aby jednu hranu nesdílely více než dva trojúhelníky. Druhé omezení se zohledňuje až dále v kontrole povrchů trojúhelníkové sítě (viz kap. 5). Pokud pro trojúhelník *T* neexistuje některý (*i*-tý) sousední trojúhelník anebo ho nelze jednoznačně určit, musí být v trojúhelníku *T* na místě pro *i*-tý sousední trojúhelník uložen speciální hodnota odkazu oznamující „tento sousední trojúhelník neexistuje !“ V naší implementaci byla pro tyto účely zvolena hodnota –1, která se po uložení do neznaménkové celočíselné proměnné převede na největší možnou hodnotu na daném rozsahu (pro námi zvolené 4 byty to tedy bude 2^{32}-1), a pokud se nebude zpracovávat model s počtem trojúhelníků větším $4 \cdot 10^{10}$ (taková data nebyla při vytváření této diplomové práce k dispozici) nedojde k žádné kolizi způsobené využitím této hodnoty.
Nyní již přistoupíme k popisu samotné implementace této metody.

Implementace metody vytváření informace o sousednostech

Pro implementaci tedy byla vybrána metoda řazení pomocí košů (viz kap. 3.2), která využívá toho, že po provedení redukce duplicitních vrcholů je ve všech trojúhelnících modelu obsahujících stejný vrchol uveden stejný index tohoto vrcholu. Pokud nelze duplicitní vrcholy z nějakých důvodů (např. nedostatek paměti) redukovat, nelze ani vytvořit informace o sousednostech trojúhelníků.

Nejprve se pro zpracovávaný vrchol V vytvoří koš trojúhelníků, které tento vrchol V obsahují. Tento koš je zde reprezentován pomocí 3 vektorů o pevné velikosti 4096 položek. Vektory jsou statické proto, že před určením sousedností – tedy ve chvíli, kdy se vektory mají vytvořit – nejsme schopni říci největší počet trojúhelníků sdílejících jeden vrchol. Víme jen, že průměrný počet těchto trojúhelníků je 6 (viz hodnota q_{avg} v kap. 3.2), avšak největší počet může být mnohem vyšší. Počet položek tedy byl nastaven na hodnotu, která s jistotou zabrání možnému přetečení rozsahu vektorů.
První z těchto vektorů je vektor pojmenovaný *vdwSecondVertexOfEdge*. Do tohoto vektoru se uloží odkazy na všechny vrcholy U_i mající s vrcholem V společnou hranu. Pro každý z těchto vrcholů si uložíme do druhého vektoru, který je pojmenovaný *vdwTrianIndex*, odkaz na trojúhelník, z nějž byl vrchol U_i vložen. Do třetího vektoru – *viVertexInTrian* – si uložíme, zda je vrchol U_i ve svém trojúhelníku uložen jako jeho první, druhý nebo třetí vrchol. Tento vektor je zde z důvodu, aby se později při ukládání informace o sousednostech nemuselo znovu zjišťovat, na které pozici je vrchol U_i ve svém trojúhelníku.

Dále se použijí tři pomocné vektory (*ptiTrianNeighBoursTemp*), které jsou pracovní kopií vektorů pro uložení informace o sousednostech (viz popis datových struktur - kap. 9). Do vektorů pro uložení informace o sousednostech se budou zapisovat až teprve výsledné odkazy. Pomocí vektorů *ptiTrianNeighBoursTemp* se pro každý vrchol vytvoří neuspořádaný zřetězený seznam trojúhelníků, které mají tento vrchol společný. K vytvoření tohoto zřetězeného seznamu využijeme ještě vektor *pvftFirstTrianOfVertex*, neboli vektor odkazů na první trojúhelník, v němž se vyskytuje daný vrchol. Tento vektor svojí velikostí odpovídá stávajícímu počtu vrcholů v modelu a jeho položky tvoří struktura obsahující odkaz na první trojúhelník vrcholu a pozici, na které je vrchol v tomto trojúhelníku uložen.

Zřetězené seznamy trojúhelníků se nyní vytvoří takto (S_i – odkaz na sousední trojúhelník pro vrchol V_i):
-- složitost této části je $O(N)$, je-li N počet trojúhelníků
Pro každý vrchol V_i každého trojúhelníku T_i
Je-li T_i prvním trojúhelníkem V_i
Do vektoru *pvftFirstTrianOfVertex* si na pozici pro V_i ulož odkaz na T_i a pozici V_i v T_i
Do T_i si jako S_i ulož odkaz na sebe sama (T_i) – zajistíme uzavření seznamu
Není-li T_i prvním trojúhelníkem V_i
Zařaď T_i do zřetězeného seznamu tímto způsobem
V T_i si jako S_i ulož odkaz, který je v prvním trojúhelníku V_i (viz vektor *pvftFirstTrianOfVertex*) uložen jako S_i
V prvním trojúhelníku V_i jako S_i ulož odkaz na T_i
-- T_i se tedy vloží do seznamu mezi první trojúhelník seznamu a naposledy vkládaný trojúhelník

Nyní máme připraveno všechno potřebné pro vytvoření košů (V_j a V_k jsou zbylé dva vrcholy v trojúhelníku T_i) a jejich následným zpracováním určení informace o sousednostech:
Pro každý vrchol V_i
-- vytvoření koše pro vrchol V_i - složitost této části je $O(L)$, je-li L průměrný počet trojúhelníků v seznamu
Pro každý trojúhelník T_i ze zřetězeného seznamu trojúhelníků
-- zpracování hrany V_iV_j
Do vektoru *vdwSecondVertexOfEdge* ulož odkaz na vrchol V_j
Do vektoru *vdwTrianIndex* ulož odkaz na T_i
Do vektoru *viVertexInTrian* ulož pozici vrcholu V_j v trojúhelníku T_i
-- zpracování hrany V_iV_k
Do vektoru *vdwSecondVertexOfEdge* ulož odkaz na vrchol V_k
Do vektoru *vdwTrianIndex* ulož odkaz na T_i
Do vektoru *viVertexInTrian* ulož pozici vrcholu V_k v trojúhelníku T_i

-- uspořádání koše podle hodnot ve vektoru *vdwSecondVertexOfEdge*
-- Byla použita metoda řazení zaměňováním (angl. *BubbleSort*) [4]
-- složitost je $O(L^2)$, je-li L průměrný počet trojúhelníků v seznamu

-- určení sousedností trojúhelníků - složitost je $O(L)$, je-li L průměrný počet trojúhelníků v seznamu
-- informace o sousednostech se již zapisují přímo do vektorů pro uložení informace o sousednostech
Pro každou položku P_i v koši
-- hledání stejných hran v koši
Je-li ve vektoru *vdwSecondVertexOfEdge* za P_i pouze jedna položka se stejnou hodnotou
-- hrana je sdílena právě dvěma trojúhelníky
S využitím vektorů *vdwTrianIndex*, *viVertexInTrian* ulož do obou trojúhelníků vzájemnou sousednost
Není-li ve vektoru *vdwSecondVertexOfEdge* za P_i žádná položka se stejnou hodnotou
anebo je-li tam více než jedna položka se stejnou hodnotou
-- hrana se nachází pouze v jednom trojúhelníku anebo je naopak sdílena více než dvěma trojúhelníky
S využitím vektorů *vdwTrianIndex*, *viVertexInTrian* ulož do všech trojúhelníků z koše obsahujících tuto hranu jako odkaz na sousední trojúhelník pro vrchol *Vi* značku „neexistující sousední trojúhelník“

Ze složitostí uvedených u jednotlivých částí metody plyne, že celková složitost vytvoření a zpracování všech košů je $O(N + M \cdot L^2)$, kde M je stávající počet vrcholů a L průměrný počet trojúhelníků sdílejících stejný vrchol. Víme, že průměrný počet trojúhelníků sdílejících stejný vrchol je 6 (viz hodnota q_{avg} v kap. 3.2) a tedy, že L << M, hovoříme-li o modelech složených z trojúhelníků o počtu řádově vyšším 10^6.

12 Určení počtu a typů různých povrchů

12.1 Datové struktury použité pro povrchy

Nejprve si popíšeme jaké struktury jsou využity k uložení a označení jednotlivých povrchů. Pro uložení informací o každém nalezeném povrchu byla použita struktura s těmito položkami:

- *identifikátor povrchu* – neznaménkové celé číslo na rozsahu 4 bytů. Tak velký rozsah se uplatní ve chvíli, kdy bude každý trojúhelník nebo nějaká menší skupinka trojúhelníků tvořit samostatný povrch. K tomu může dojít, jestliže budou vrcholy v trojúhelnících zadány s odchylkou a při redukci duplicitních vrcholů dojde k „roztržení“ modelu
- *typ povrchu* – určuje, zda je povrch uzavřený (bez děr) nebo otevřený (s dírami)
- *orientace trojúhelníků povrchu* – tento příznak určuje, zda jsou trojúhelníky povrchu orientovány proti nebo po směru hodinových ručiček (viz kap. 6). Tato položka se inicializuje na hodnotu označující orientaci proti směru hodinových ručiček
- *povrch zorientován* – příznak, že všechny trojúhelníky povrchu mají (ověřenou) shodnou orientaci. Tento příznak se využije dále při nastavení stejné orientace pro všechny trojúhelníky povrchu (viz kap. 13)
- *povrch vybrán* – příznak, že povrch je vybrán pro provedení operace (např. pro zobrazení vybraných povrchů).
- *odkaz na první trojúhelník povrchu* – slouží k „napojení“ povrchu na jeho trojúhelníky
- *celkový počet děr, vrcholů, hran a trojúhelníků povrchu* – tyto počty se uplatní například při kontrole konzistence modelu pomocí *Eulerovy rovnice* (viz kap. 5)
- *počet vrcholů, hran a trojúhelníků povrchu, které incidují s dírou v povrchu* – také tyto počty se uplatní například při kontrole konzistence modelu pomocí *Eulerovy rovnice*, ale také při odstraňování děr z povrchů
- *barva povrchu* – využije se při zobrazování modelu, aby bylo možné zobrazit každý povrch jinou barvou
- *souřadnice obalového kvádru povrchu* – využití pro zjišťování, zda nedochází k průniku povrchů, apod.

Všechny povrchy jsou uloženy v seznamu povrchů. Ten je implementován jako zřetězený dynamický seznam vytvořený ze šablony CList poskytnuté knihovnou *Microsoft Foundation Classes*[3]. Vzhledem k tomu, že při opravě chyb v modelu může dojít ke sloučení dvou či více povrchů, je nutné mít možnost jednoduše položku povrchu ze seznamu odstranit. Z tohoto důvodu byl zvolen datový typ zřetězený dynamický seznam.

12.2 Popis metody pro určení povrchů v modelu

Než se začnou zjišťovat povrchy v modelu, musí se samozřejmě nejprve určit sousednosti trojúhelníků. Jinak by bylo nalezeno tolik povrchů, kolik je trojúhelníků a to je samozřejmě nežádoucí a hlavně bezvýznamné.

Nejprve se provede inicializace seznamu povrchů, pole identifikátorů povrchů jednotlivých trojúhelníků a všech čítačů vrcholů, hran, atd. Pole identifikátorů povrchů jednotlivých trojúhelníků se vyplní nulami, což je využito jako příznak, že daný trojúhelník ještě nemá přiřazen povrch.

Dále se v cyklu prochází pole trojúhelníků a pro každý trojúhelník T_i se provede toto:
Má-li již trojúhelník T_i nebo některý z jeho (existujících) sousedů S_1, S_2, S_3 přiřazen identifikátor povrchu
Urči nejmenší hodnotu h z těchto identifikátorů (pozor na nulu, která znamená „povrch dosud nepřiřazen“)
Přiřaď všem trojúhelníkům T_i, S_1, S_2 a S_3 jako identifikátor povrchu hodnotu h
Nemá-li ani jeden z trojúhelníků T_i, S_1, S_2 a S_3 ještě přiřazen identifikátor povrchu
Přiřaď všem trojúhelníkům T_i, S_1, S_2 a S_3 dosud nepřiřazený identifikátor (první možná hodnota je 1)

Tento cyklus se opakuje vždy znovu, pokud při posledním průchodu došlo v nějakém trojúhelníku ke změně identifikátoru jeho povrchu. Kromě toho se vždy zapamatuje trojúhelník T_{min} s nejmenším a T_{max} s největším indexem, u kterých došlo k této změně a další iterace cyklu se provede od trojúhelníku T_{min} do trojúhelníku T_{max} včetně. Opakujícím se prohledáváním pole trojúhelníků a přiřazováním vždy nejmenší hodnoty identifikátoru povrchu dosáhneme toho, že po několika iteracích bude všem trojúhelníkům jednoho povrchu přiřazen stejný identifikátor. A bude to ten identifikátor, který byl přiřazen jako první některému z trojúhelníků povrchu.

Nyní je nutné uspořádat identifikátory do souvislé řady, neboť právě přiřazováním vždy nejmenší hodnoty identifikátorů, došlo k vyřazení některých hodnot. Identifikátory povrchů jsme doposud ukládali k jednotlivým trojúhelníkům a seznam povrchů je tedy dosud prázdný. Nyní ho naplníme prvky pro jednotlivé povrchy a zároveň provedeme uspořádání přiřazených identifikátorů do souvislé řady, tak aby pro N povrchů byly použity identifikátory 1 .. N. Zatím však budeme identifikátory měnit pouze v poli identifikátorů povrchů jednotlivých trojúhelníků a do seznamu povrchů budeme ukládat povrchy ještě s nezměněnými identifikátory.
V následujícím popisu budeme seznam povrchů označovat jako *SP*.

Pro všechny trojúhelníky T_i
I je identifikátor povrchu trojúhelníku T_i
Existuje-li již v *SP* povrch s identifikátorem I
K trojúhelníku T_i přiřaď identifikátor povrchu, který odpovídá pozici jeho povrchu v *SP*
Neexistuje-li ještě v *SP* povrch s identifikátorem I
Přidej na konec *SP* položku povrchu s identifikátorem I, pro který je T_i prvním trojúhelníkem
Aktualizuj položky (kromě počtu vrcholů) povrchu v *SP* podle odpovídajících hodnot z trojúhelníku T_i

Tak dosáhneme seřazení identifikátorů do souvislé posloupnosti, zatím však jen v poli identifikátorů povrchů. Kromě toho u každého povrchu máme zjištěny celkové počty jeho trojúhelníků a hran a počty trojúhelníků a hran, které incidují s dírou, tedy takové, u nichž nebyl nalezen nebo určen sousední trojúhelník. Každý povrch, který obsahuje trojúhelník(y) incidující s dírou, označíme jako neuzavřený, ostatní povrchy jsou označeny jako uzavřené. Dále pro každý povrch známe pozici a velikost obalového kvádru, což se nám bude hodit při zjišťování, zda se některé povrchy neprotínají.

[3] knihovna tříd poskytovaná firmou Microsoft k vývojovému prostředí Microsoft Visual C++

Teď musíme ještě seřadit do souvislé posloupnosti identifikátory uložené v seznamu povrchů. Také je třeba určit pro každý povrch počty všech vrcholů a počty vrcholů incidujících s dírou. Za tímto účelem si vytvoříme pomocný vektor příznaků, do nějž budeme ukládat hodnoty značící, že vrchol je již v daném povrchu započítán v celkovém počtu vrcholů (hodnota 1) a že je již v daném povrchu započítán i jako vrchol incidující s dírou (hodnota 2). Použití tohoto vektoru je nutné, protože z každého trojúhelníku máme odkaz pouze na sousední trojúhelníky. S daným vrcholem však inciduje ve většině případů více trojúhelníků a my bychom pak museli složitě všechny tyto trojúhelníky procházet a zjišťovat, zda již byly zpracovány a tedy daný vrchol je již započítán či nikoliv. S použitím tohoto pomocného vektoru si vystačíme s jednou příp. dvěma operacemi porovnání.
Dále každý povrch, který obsahuje trojúhelník(y) incidující s dírou, přičteme k počtu neuzavřených povrchů.
Seznam povrchů budeme v následujícím popisu opět označovat jako *SP*. Pomocný vektor příznaků pro označení, zda vrchol je v povrchu již započítán či nikoliv, budeme označovat jako *VP*.

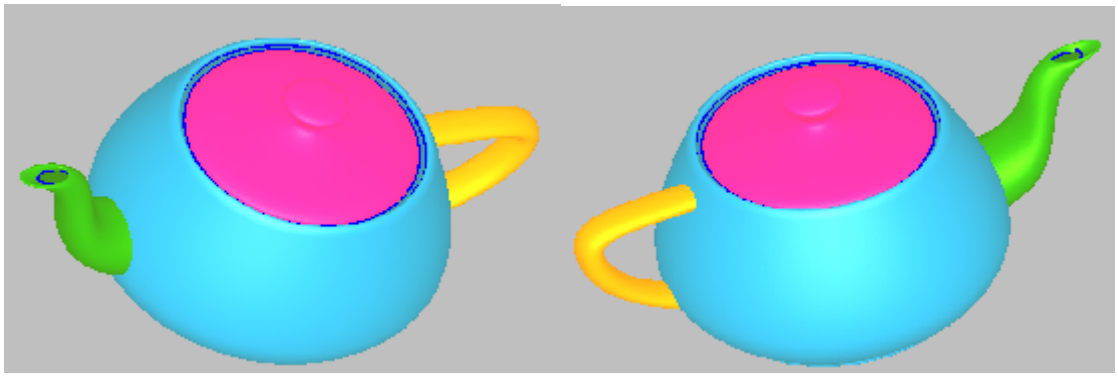
Pro každý povrch P_i z *SP*
Jako identifikátor povrchu P_i si ulož hodnotu odpovídající jeho pozici v *SP*
Inicializuj *VP*

Pro každý trojúhelník T_i – začni od prvního trojúhelníku povrchu P_i
Patří-li trojúhelník T_i do povrchu P_i
Započítej do celkového počtu vrcholů v povrchu všechny vrcholy T_i, které ještě započítány nejsou a označ je,
že jsou do tohoto počtu již započítány (do *VP* ulož 1)
-- nyní se bude zjišťovat, zda vrcholy trojúhelníku T_i incidují s dírou či nikoliv
Pokud neexistuje 2. nebo 3. sousední trojúhelník
První vrchol T_i inciduje s dírou - započítej ho do počtu těchto vrcholů (pokud ještě není) a označ ho, že již mezi tyto vrcholy započítán je (do *VP* ulož 2)
Pokud neexistuje 1. nebo 3. sousední trojúhelník
Druhý vrchol T_i inciduje s dírou - započítej ho do počtu těchto vrcholů (pokud ještě není) a označ ho, že již mezi tyto vrcholy započítán je (do *VP* ulož 2)
Pokud neexistuje 1. nebo 2. sousední trojúhelník
Třetí vrchol T_i inciduje s dírou - započítej ho do počtu těchto vrcholů (pokud ještě není) a označ ho, že již mezi tyto vrcholy započítán je (do *VP* ulož 2)

Povrchu P_i nyní přiřaď první dosud nepřidělený identifikátor

Tím máme vytvořeny informace o každém povrchu, který se v modelu nachází.

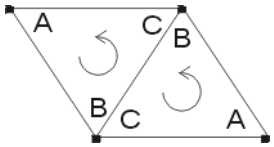
Na obr. 12.1 vidíme model ze souboru *Teapot.stl*, u něhož byly při vyhledávání povrchů správně rozpoznány čtyři různé. Každý povrch je zobrazen jinou barvou a protože ani jeden není uzavřený jsou, zde také zvýrazněny okraje děr v každém z nich.



Obr. 12.1 Povrchy v modelu Teapot.stl

13 Nastavení požadované orientace trojúhelníků v trojúhelníkové síti

Shodnou orientaci trojúhelníků zajistíme na základě sousednosti trojúhelníků.



Obr. 13.1 Shodná orientace trojúhelníků

Jak vidíme z obr. 13.1, dva trojúhelníky jsou shodně orientovány tehdy, pokud jsou vrcholy, ležící na jejich společné hraně, v obou uvedeny v opačném pořadí. Výhoda trojúhelníků v tomto případě spočívá v tom, že pro získaní shodné orientace dvou sousedních trojúhelníků, jejichž stávající orientace je rozdílná, stačí prohodit libovolné dva vrcholy v jednom z nich (např. u čtverce už bychom museli brát ohled na to, které dva vrcholy prohodit).

Abychom zjistili, zda mají dva sousední trojúhelníky shodnou orientaci, musíme zjistit, zda jsou vrcholy tvořící jejich společnou hranu v obou trojúhelnících uvedeny v opačném pořadí. Např. pro trojúhelník T a jeho první sousední trojúhelník S se musí porovnat vrcholy, které jsou v trojúhelníku T jako druhý a třetí, neboť tyto dva vrcholy jsou oběma trojúhelníky sdíleny. Je-li U_2 a U_3 druhý a třetí vrchol trojúhelníku T a V_1, V_2 a V_3 první, druhý a třetí vrchol trojúhelníku S, pak zjišťování shodnosti orientací vypadá takto (předpokládáme, že trojúhelník S se snažíme přizpůsobit orientací trojúhelníku T):

Je-li U_2 shodný s V_1
Vzhledem k tomu, že v T je vrchol U_3 uveden hned za vrcholem U_2, musí být v S uveden vrchol odpovídající
vrcholu U_3 hned před vrcholem V_1, musí to tedy být vrchol V_3. Proto, je-li s vrcholem U_3 shodný
vrchol V_2, musí dojít ke změně orientace trojúhelníku S
Je-li U_2 shodný s V_2
Vzhledem k tomu, že v T je vrchol U_3 uveden za hned vrcholem U_2, musí být v S uveden vrchol odpovídající
vrcholu U_3 hned před vrcholem V_2, musí to tedy být vrchol V_1. Proto, je-li s vrcholem U_3 shodný
vrchol V_3, musí dojít ke změně orientace trojúhelníku S

Je-li U_2 shodný s V_3
Vzhledem k tomu, že v T je vrchol U_3 uveden za hned vrcholem U_2, musí být v S uveden vrchol odpovídající
vrcholu U_3 hned před vrcholem V_3, musí to tedy být vrchol V_2. Proto, je-li s vrcholem U_3 shodný
vrchol V_1, musí dojít ke změně orientace trojúhelníku S

Pro druhý a třetí sousední trojúhelník je porovnání analogické. Ať již byla orientace trojúhelníku S měněna, nyní je požadovaného typu a my si trojúhelník S označíme jako zorientovaný.

Otočení orientace trojúhelníku se provádí tak, že se prohodí odkazy na první a druhý sousední trojúhelník a odkazy na první a druhý vrchol trojúhelníku. Jak bylo uvedeno výše, u trojúhelníku nezáleží na tom, které dva vrcholy se prohodí, aby se získala opačná orientace, proto je vždy prohazován první a druhý vrchol trojúhelníku.

Metoda pro nastavení shodné orientace trojúhelníků v trojúhelníkové síti je implementována tak, že orientuje pouze trojúhelníky jednoho povrchu, jehož identifikátor P dostane na vstupu. To nám umožní určit orientaci pro každý povrch zvlášť, neboť ne vždy budou všechny povrchy sami o sobě orientovány stejně. Kromě identifikátoru povrchu dostává metoda na vstupu požadovaný typ orientace - tedy po nebo proti směru hodinových ručiček.

Pro uložení příznaků, že trojúhelník ještě není orientován, je v implementaci této metody použit vektor, v němž je pro každý vrchol vyhrazena položka typu *BOOLEAN*. Hodnota *TRUE* znamená, že daný trojúhelník ještě orientován není, lépe řečeno jeho orientaci dosud neznáme.

Pokud nemá daný povrch nastaven příznak, že již má nastavenu orientaci, pak se provede zorientování trojúhelníků tohoto povrchu způsobem, který je popsán na následujících řádcích.

Nejprve se podle typu požadované orientace zjišťuje, zda je správně orientován první trojúhelník povrchu, neboť podle něho se následně zorientují všechny trojúhelníky. Pokud nemá požadovaný typ orientace, jeho orientace se otočí.

Dále se v cyklu prochází pole trojúhelníků a pro každý trojúhelník T_i patřící do povrchu P se provede toto:

Není-li trojúhelník T_i ještě zorientován
Existuje-li alespoň jeden sousední trojúhelník S_i, který je již zorientován
Zorientuj trojúhelník T_i shodně s trojúhelníkem S_i

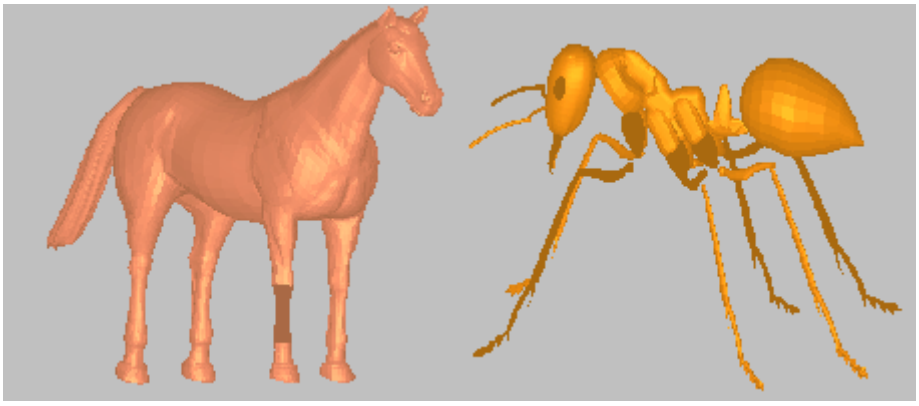
Je-li trojúhelník T_i nyní zorientován
Zorientuj všechny dosud nezorientované (existující) sousední trojúhelníky S_i podle trojúhelníku T_i

Tento cyklus se opakuje vždy znovu, pokud při posledním průchodu došlo ke změně orientace nějakého trojúhelníku. Kromě toho se vždy zapamatuje trojúhelník T_{min} s nejmenším a T_{max} s největším indexem, u kterých došlo k této změně a další iterace cyklu se provede od trojúhelníku T_{min} do trojúhelníku T_{max} včetně. Opakujícím se prohledáváním pole trojúhelníků a postupným zorientováním trojúhelníků podle jejich již zorientovaných sousedů dosáhneme toho, že po několika iteracích budou mít všechny trojúhelníky zvoleného povrchu stejnou (požadovanou) orientaci.

Nakonec si k danému povrchu uložíme typ právě nastavené orientace a nastavíme příznak, že všechny trojúhelníky povrchu jsou zorientovány.

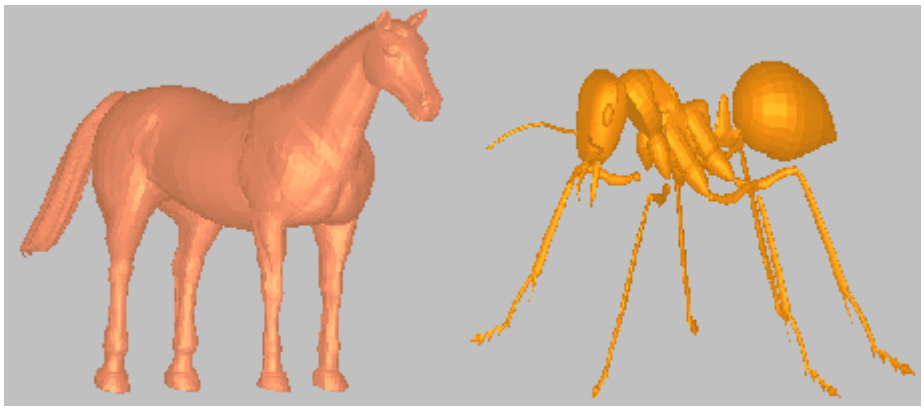
Takto se tedy zpracuje povrch, který ještě není zorientován. Má-li se však nastavit požadovaná orientace u povrchu P, jehož trojúhelníky již mají nastavenu shodnou orientaci (viz příznak *povrch zorientován* v datové struktuře pro povrchy – kap. 12.1), pak se projde celé pole trojúhelníků pouze jednou a každému trojúhelníku, který patří do povrchu P, se změní jeho orientace. To se provede ovšem pouze v případě, že je požadována opačná orientace, než kterou mají trojúhelníky povrchu.

13.1 Dosažené výsledky



Obr. 13.2 Opačné orientace trojúhelníků modelu

Na obr. 13.2 vidíme modely *Horse.stl* a *Ant.stl*, jedná se o model koně a mravence. Tyto modely jsou dodávány k aplikaci 3D Studio MAX jako vzorové a při jejich exportu do formátu STL došlo k chybnému nastavení orientací trojúhelníků. U modelu koně je vidět na pravé přední noze část, která má odvrácený normálový vektor. V modelu mravence je takových částí více. Na obr. 12.3 již vidíme, jak byl naším algoritmem tento nedostatek u obou modelů napraven.



Obr. 13.3 Shodné orientace všech trojúhelníků modelu

14 Výpočet normálových vektorů ve vrcholech

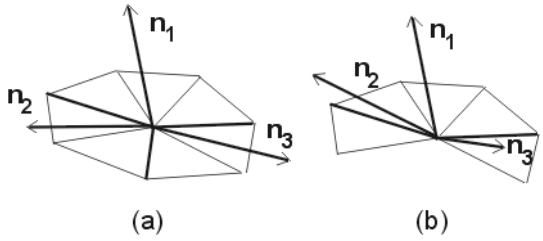
Pro výpočet normálových vektorů ve vrcholech trojúhelníkové sítě byla ze zmíněných metod (viz kap. 7) vybrána metoda průměrného normálového vektoru a metoda vyhledání ostrých hran v trsu. Metoda průměrného normálového vektoru byla zvolena proto, že struktury MVE umožňují uložení pouze jednoho normálového vektoru ke každému vrcholu, což je také výstupem této metody. Metoda vyhledání ostré hrany v trsu byla zvolena proto, že umožňuje lepší výsledný dojem ze zobrazeného modelu.

Metoda vyhledání ostré hrany v trsu

Zda je hrana ostrá se určuje na základě hodnoty funkce kosinus úhlu, který svírají dva trojúhelníky sousedící právě přes tuto hranu. Tato hodnota k se vypočte jako vektorový součin normálových vektorů $\boldsymbol{u}$, $\boldsymbol{v}$ těchto trojúhelníků:

$$k = u_x \cdot v_x + u_y \cdot v_y + u_z \cdot v_z \qquad (14.1)$$

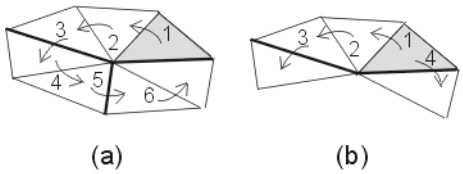
Pokud je vypočtená hodnota k menší než prahová hodnota k_{prah}, pak je hrana mezi těmito trojúhelníky považována za ostrou. Ostré hrany nalezené v rámci jednoho trsu, tento trs rozdělí a normálový vektor v daném vrcholu je pro každou část trsu mezi dvěma ostrými hranami vypočten jako průměrná hodnota z normálových vektorů trojúhelníků, které se v dané části trsu nacházejí (viz obr. 14.1).
Je patrné, že hodnotou k_{prah} lze ovlivnit výsledné normálové vektory a tím i výsledný dojem při zobrazení modelu. Vliv hodnoty k_{prah} je vidět na obrázcích uvedených na konci této kapitoly.



Obr. 14.1 Normálové vektory ve vrcholu pro různé části trsu

(a) Uzavřený trs
(b) Otevřený trs

Trs okolo zpracovávaného vrcholu V se prochází tak, že se z trojúhelníku T_i přejde do dalšího trojúhelníku T_j přes opačnou hranu incidující s vrcholem V, než přes kterou jsme se do trojúhelníku T_i dostali. To se provádí tak dlouho, dokud se nevrátíme zpět do trojúhelníku, ve kterém jsme začali, v případě, že je trs uzavřený, anebo dokud nedojdeme do trojúhelníku, který nemá potřebný sousední trojúhelník, v případě, že je trs otevřený. U otevřeného trsu je třeba z prvního trojúhelníku začít prohledávat znovu, ale opačným směrem, abychom prošli celý trs (viz obr. 14.2).



Obr. 14.2 Způsob procházení trsem

(a) Uzavřený trs
(b) Otevřený trs

Při procházení trsu se může stát, že začneme v trojúhelníku, ve kterém nezjistíme ostrou hranu a který tedy leží uvnitř části trsu. S touto možností musíme počítat a při nalezení první ostré hrany je třeba si všechny informace o první části trsu uchovat. Poté, co obejdeme celý uzavřený trs a vrátíme se zpět do jeho první části, tyto hodnoty použijeme pro výpočet normálového vektoru této části trsu. U otevřeného trsu tyto

hodnoty použijeme tehdy, když se dojde na jeden jeho konec. Pak se uchovanými hodnotami inicializují potřebné informace o dané části trsu a začínáme znovu z prvního trojúhelníku.

Co tedy potřebujeme o dané části trsu vědět, abychom byli schopni vypočítat jeho normálový vektor ? Potřebujeme si uchovat odkazy na trojúhelníky dané části trsu a pozici vrcholu zpracovávaného vrcholu V v každém z těchto trojúhelníků, což nám později usnadní ukládání vypočítaného normálového vektoru. Za těmito účely jsou použity vektory *vdwTrianIndexOfFirstSubStrip* a *viVertexInTrianOfFirstSubStrip*, které jsou určeny pro uchování potřebných informací z první části trsu a vektory *vdwTrianIndexOfNextSubStrip* a *viVertexInTrianOfNextSubStrip*, které se používají při zpracování jednotlivých trsů. Dále potřebujeme samozřejmě znát počet trojúhelníků v dané části trsu, pro výpočet průměru z normálových vektorů, jejichž jednotlivé složky sčítáme do proměnných *glfNormalX*, *glfNormalY*, *glfNormalZ*. Vzhledem k tomu, že do pole vrcholů přistupujeme postupně přes odkazy z jednotlivých trojúhelníků, potřebujeme ještě vektory *pbVertexNormalToDo*, do kterých uložíme příznak, že v daném trojúhelníku byl pro vrchol na dané pozici normálový vektor již vypočítán. Tím zajistíme, že normálový vektor bude pro každý vrchol počítán pouze při jeho prvním výskytu.

Celá metoda tedy pracuje následovně:
Pro každý vrchol V_i každého trojúhelníku T_i
Nebyl-li pro V_i ještě počítán normálový vektor
Procházej trs jedním směrem dokud nezjistíš ostrou hranu, ukládej si odkazy na trojúhelníky T_j, pozice vrcholu V_i
v každém T_j a složky jejich normálových vektorů přičítej ke složkám počítaného normálového vektoru

Nebyla-li ostrá hrana nalezena v celém trsu
Vypočti průměrnou hodnotu z normálových vektorů všech trojúhelníků trsu, ulož ji a přejdi na další vrchol V_i

Byla-li nalezena ostrá hrana
Jde-li o první nalezenou ostrou hranu v trsu
Zálohuj všechny potřebné informace pro pozdější využití
Nejde-li o první nalezenou ostrou hranu v trsu
Vypočti na základě zjištěných informací normálový vektor a ulož jej

Bylo-li dosaženo krajního trojúhelníku v otevřeném trsu
Nejedná-li se již o druhou koncovou hranu otevřeného trsu
Nejsme-li stále v první části trsu
Vypočti na základě zjištěných informací normálový vektor a ulož jej
Obnov informace první části trsu
Vrať se zpět do prvního trojúhelníku, a pokračuj v procházení trsu opačným směrem
Jedná-li se již o druhou koncovou hranu otevřeného trsu
Vypočti na základě zjištěných informací normálový vektor a ulož jej

Bylo-li dosaženo posledního trojúhelníku v uzavřeném trsu
K hodnotám pro danou část trsu přidej i hodnoty uchované z první části trsu a vypočti normálový vektor a ulož jej

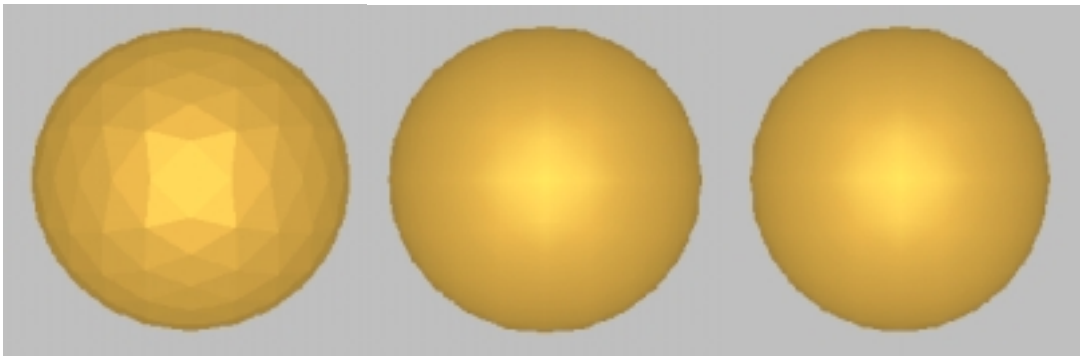
Takovýmto způsobem tedy vypočteme normálové vektory ve vrcholech trojúhelníkové sítě metodou hledání ostré hrany.

Průměrný normálový vektor

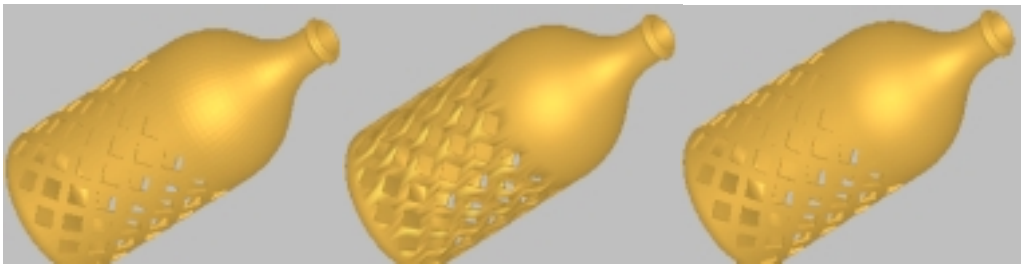
V této metodě se jednoduše projde pole trojúhelníků a pro každý vrchol V_i každého trojúhelníku T_i se ke složkám normálového vektoru vrcholu V_i přičtou složky normálového vektoru trojúhelníku T_i. Zároveň si ke každému vrcholu ukládáme počet trojúhelníků. Pak projdeme pole vrcholů a pro každý vrchol V_i vypočteme jeho normálový vektor vydělením jeho jednotlivých složek počtem trojúhelníků, v nichž je vrchol obsažen.

14.1 Dosažené výsledky

Nyní porovnáme obě metody zvolené pro implementaci. Pro srovnání si uvedeme i výsledky získané konstantním stínováním. Pro porovnání metod byly zvoleny modely koule a model lahve s ozdobnými otvory.



Obr. 14.3 Stínování povrchu modelu koule
Vlevo: konstantní stínování
Vpravo: stínování metodu Gouraud – průměrný normálový vektor, hledání ostrých hran



Obr. 14.4 Stínování povrchu modelu lahve
Vlevo: konstantní stínování
Vpravo: stínování metodu Gouraud – průměrný normálový vektor, hledání ostrých hran

U modelu krychle se obě metody chovají stejně, neboť tento model neobsahuje ostré hrany a proto se v obou případech normálový vektor počítá jako průměr z normálových vektorů všech trojúhelníků incidujících s daným vrcholem. Naopak u modelu lahve jsou již vidět nevýhody metody pro výpočet normálového vektoru jako průměru z normálových vektorů s vrcholem incidujících trojúhelníků. Zejména v okolí ozdobných otvorů, kde jsou mezi trojúhelníky úhly 90° a tedy by měla být brána hrana mezi nimi v úvahu jako skutečná (ostrá).

15 Závěr

V současné době jsou na základě znalostí uvedených v teoretické části implementovány a ověřeny metody pro redukci duplicitních vrcholů a pro zjišťování sousedností v trojúhelníkové síti. Dále jsou to metody pro nalezení různých povrchů v trojúhelníkové síti a zajištění jejich správné orientace. Podařilo se také implementovat metody pro výpočet normálových vektorů ve vrcholech trojúhelníkové sítě a na zákadě dosažených výsledků ověřit, které metody jsou vhodné pro účely zobrazování trojúhelníkových sítí.

V rámci této diplomové práce se zejména podařilo dokázat, že hashovací funkce lze efektivně využít při prohledávání množiny vrcholů. Navržený tvar hashovací funkce během testovaní vykazoval velmi dobré vlastnosti a délky shluků na pozicích hashovací tabulky byly nejvýše v řádu 10.

Pro další práci tedy ještě zbývá výběr, implementace a ověření metod pro opravu chyb ve struktuře trojúhelníkové sítě.

16 Použitá literatura

[1] Glassner,A.:Computing Surface Normals for 3D Models, Graphic Gems I, Academic Press, Inc., 1998

[2] Glassner,A.:Building Vertex Normals from an Unstructured Polygon List, Graphic Gems IV, str.60 - 73, Academic Press, Inc., 1994

[3] Kofrhage,R.R., Gibbs,N.E.: Principles of Data Structures and Algorithms with Pascal, Wm. C. Brown Publishers, 1987

[4] Ježek, K., Klečková, J., Ledvina, J., Počítače a programování, ZČU Plzeň, 1995

[5] Žára, J., Beneš, B., Felkel, P., Moderní počítačová grafika, Computer Press , 1998

[6] Stl Format Description, http://www.sdsc.edu/tmf/Stl-specs/stl.html

[7] Tele-Manufacturing Facility Home Page, http://tmf.sdsc.edu/

[8] Rapid Prototyping Electronic Mailing List, http://ltk.hut.fi/rp-ml/

[9] Konference Algoritmy 2000, Slovensko, 15.-20. Září 2000