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

Diplomová práce

Hierarchické trojúhelníkové sítě a jejich redukce

Plzeň, 2003 Jiří Fornous

Abstract

This work is focused on simplification of triangular nets and construction of multitriangulation. The simplification is done with the help of the method of iterative contraction of an edge. Basic Quadric error metric is used for the valuation of the edges for contraction. Hierarchical triangular net is realized with the help of explicit multitriangulation. In practice, also a basic algorithm of selective refinement appears.

Obsah

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

V Plzni dne 28.5.2003
Jiří Fornous

1 Úvod

Současná doba je doba nebývalého zájmu o informace ve všech různých oblastech vědeckého výzkumu. Množství požadovaných informací se neustále zvětšuje, a to především ze dvou důvodů. Jedním z nich jsou stále dokonalejší měřící přístroje, které získávají stále větší počet dat. Druhým důvodem je stále jemnější diskretizace matematických modelů, jež opět produkuje větší a větší datové soubory. Oblastí, kde se tento informační rozmach intenzivně projevuje je právě počítačová grafika. Objemy trojrozměrných dat, která je třeba zobrazit, jsou velmi rozsáhlé a samotný hardwarový vývoj není schopný tento nárůst dostatečně kompenzovat. Je tedy třeba současně i vyvíjet adekvátní softwarové modely, které by vhodně vyvážily atributy množství, přesnost a rychlost zobrazované informace.
Názorným příkladem je tomu získávání trojrozměrných dat při scanování zemského povrchu. Nad těmito daty se provádí různé simulace průletů. Při průletu daty je většinou potřeba velmi úzce lokalizovaných dat v co nejlepší kvalitě a ještě k tomu v reálném čase. Navíc se objevuje potřeba možnosti rychlé změny lokality na větší vzdálenost a přitom mít možnost aspoň orientačně zobrazovat průběh změny. Tyto čtyři faktory současně se snaží řešit právě algoritmy multitriangulace.

2 Teoretická část

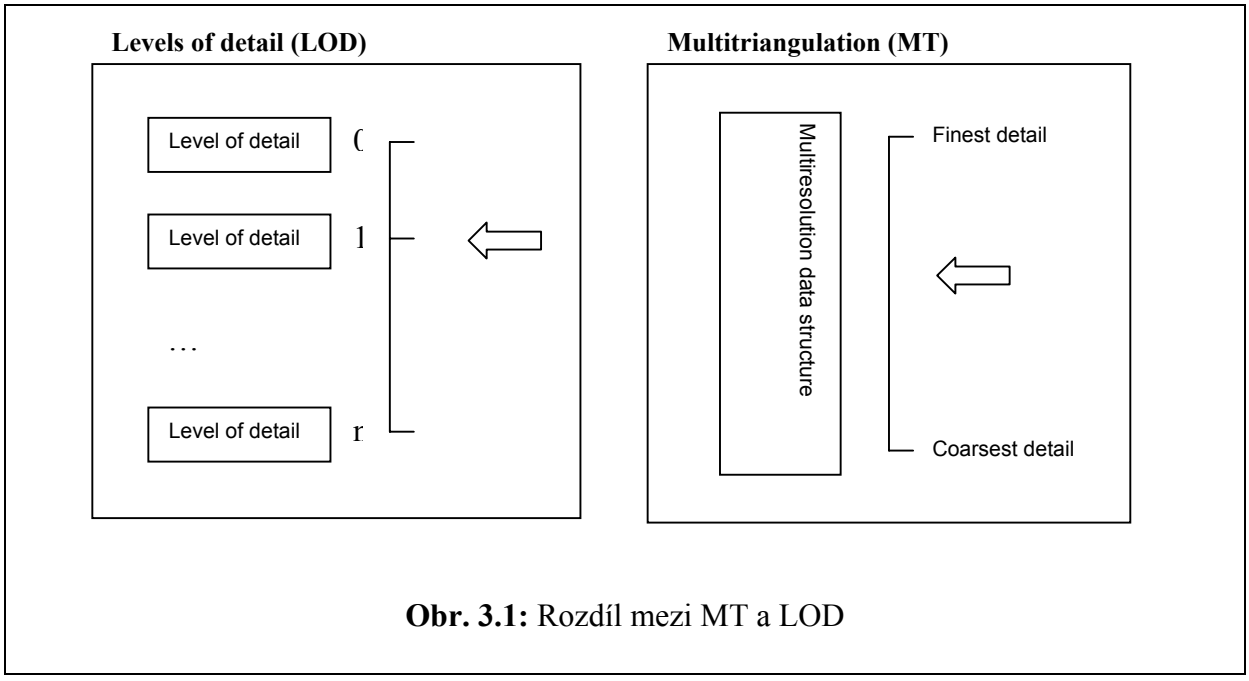
2.1 Zavedení pojmů

Zjednodušování
[simplification] – proces zjednodušování původní trojúhelníkové sítě. Vyznačuje se velkou variabilitou – existuje spousta způsobů zjednodušování trojúhelníkové sítě.

Úroveň detailu
[level of detail] – konkrétní stupeň zjednodušení (zjemnění). Může být zaznamenán samostatně nebo být zakódován v MT.

Úrovně detailu (LOD)
[levels of detail] – model založený na uchovávání na sobě nezávislých úrovní detailu jako celku. Je to určitá hierarchie úrovní detailu. Oproti MT však produkuje jen omezenou množinu různých trojúhelníkových sítí. Někdy se však nedodržuje podmínka nezávislosti úrovní a hranice mezi MT a LOD se tím pádem stává značně nejasnou.

Multitriangulace (MT)
[multitriangulation] – model založený na mnoha LOD. V modelu je často zachycený postup simplifikace do odpovídajících datových struktur. Umožňuje pohyb v těchto strukturách přes různé LOD.



Obr. 3.1: Rozdíl mezi MT a LOD

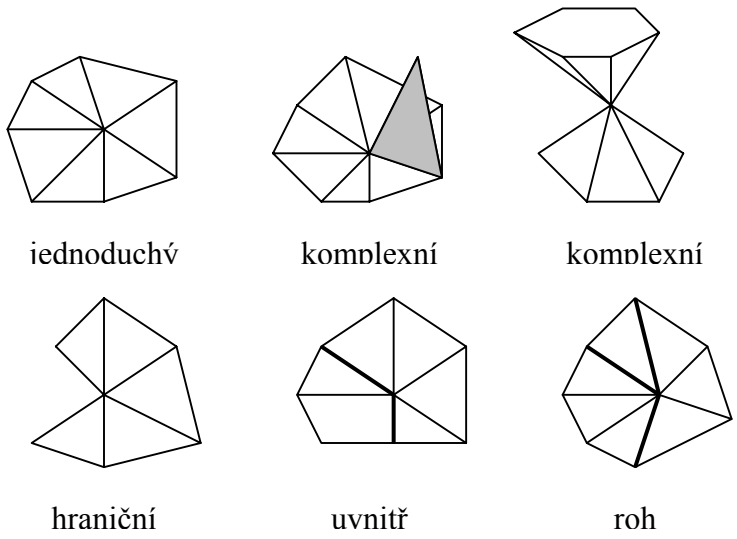
2.2 Zjednodušování
[simplification]

2.2.1 Metody založené na decimaci bodu, hrany, plošky

2.2.1.1 Decimace trojúhelníkové sítě
Metoda je založena na cyklickém procházení vrcholů trojúhelníkové sítě. Pokud vrchol splňuje decimační kriteria, je spolu s trojúhelníky, které s ním incidují odstraněn a vzniklá díra je zatriangulována. Proces se opakuje až do doby, kdy je dosaženo nějakého decimačního kriteria nebo zastavovací podmínky. Algoritmus se skládá ze tří částí:

- charakteristika lokální geometrie a topologie
- ohodnocení decimačního kriteria
- triangulace vzniklé díry

2.2.1.1.1 Charakteristika lokální geometrie a topologie
Každý bod spadá do jedné z pěti skupin podle následujícího schématu:



Obr.: Šest klasifikačních skupin používaných k výběru kandidátů na vypuštění

jednoduchý vrchol – vrchol, který obklopuje nepřerušený vějíř trojúhelníků a každá z hran vycházejících z trojúhelníku je společná pouze pro dva vrcholy
komplexní vrchol – pokud je některá hrana sdílena více než dvěma trojúhelníky nebo pokud trojúhelníky netvoří jeden kompletní vějíř
hraniční – kolem vrcholu je jeden neúplný vějíř
uvnitř hrany – pokud z vrcholu vedou jen dvě hrany (hrana je klasifikována na základě nějakého hraničního úhlu mezi sousedními trojúhelníky)
roh – z vrcholu vede více než dvě hrany nebo jen jedna

Pokud je vrchol ohodnocen jako jednoduchý, hraniční nebo je uvnitř hrany, je adeptem k vypuštění.

2.2.1.1.2 Ohodnocení decimačního kriteria

Pro jednoduché vrcholy je decimačním kriteriem vzdálenost od průměrné roviny. Pokud spadá do určené hodnoty, vrchol je možné vypustit. Hraniční vrcholy a vrcholy uvnitř hrany se schvalují k vypuštění pokud vzdálenost vrcholu od přímky tvořené koncovými body hrany nebo hranice nepřesáhne zadanou hodnotu.

2.2.1.1.3 Retriangularizace

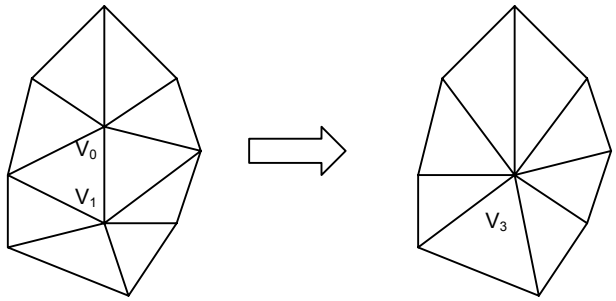
Pokud je možné retriangularizovat vzniklou díru, je tak učiněno, pokud ne, vypuštění bodu je zrušeno.

2.2.1.1.4 Zhodnocení metody

Metoda je založena na lokálním kriteriu současného meshe nikoliv meshe původního. Chyba vzniklá při aproximaci se tedy akumuluje během iteračního procesu. Výhodou algoritmu je, že podle autorů dobře udržuje ostré hrany v objektu. Existují také různé modifikace, které se snaží řešit problém globální chyby.

2.2.1.2 Kontrakce hrany za použití kvadrické chybové metriky [garl9901]

[quadric error metric]
Metoda je založená na kontrakci hrany. Hrany jsou ohodnoceny podle kvadratické vzdálenosti obou krajních bodů od všech ploch (trojúhelníků) oblasti, do které spadá. Kvadratická vzdálenost bodu má zajímavou analogii ve výpočtu bodu na izoploše kvadriky, jejíž matice je tvořena subvýsledky výpočtu již zmiňované vzdálenosti.
Při inicializaci zjednodušování se vypočtou kvadriky všech bodů původní sítě. Ohodnocení hrany se pak získá výpočtem ze součtové kvadriky obou krajních bodů. Navíc lze jednoduše určit pozici minimálního nového bodu (bodu, který leží ve středu kvadriky). Podle získaného ohodnocení se seřadí hrany do fronty adeptů.
Při výpočtu se vždy vezme hrana s nejmenším ohodnocením, kontrahuje se a přehodnotí se všechny hrany incidující s vrcholy zaniklé hrany. Takto se postupuje až do nejjednoduššího tvaru.



Obr.: Kontrakce hrany se vznikem nového bodu

2.2.1.2.1 Zhodnocení metody

Metoda je hojně využívána především pro její jednoduchou implementaci, měření globální chyby. Je poměrně rychlá a velice stabilní. Protože je založená na zjednodušování pomocí hrany, není třeba žádná zpětná retriangulace. Počet kroků je shora omezen.

2.2.2 Metody založené na shlukování

2.2.2.1 Adaptivní zjednodušování za použití shluků bodů[garl0101]

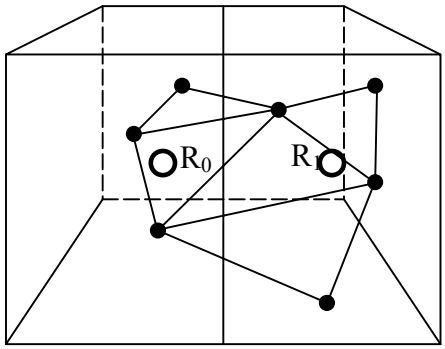
[adaptive simplification using vertex clustering]

Tato metoda používá Quadric error metric (QEM) a Dual quadric error metric(DQEM). QEM stejně jako u kontrakce hrany pomocí QEM pomáhá k nalezení bodu s nejmenší vzdáleností od skupiny rovin. Naproti tomu DQEM hledá rovinu, která má nejmenší vzdálenost od skupiny bodů. Tyto matice jsou blíže popsány v [garl0101]. V zásadě probíhá trojfázově:

- kvantizace
- konstrukce BSP stromu
- zjednodušení

2.2.2.1.1 Kvantizace

Je vytvořena pravidelná mřížka přes celý objekt (stačí pouze znalost Bounding Boxu). Za pomoci všech bodů spadajících do jednotlivých buňek je vypočtena QEM a DQEM. Z QEM se vypočte reprezentativní bod dané buňky.



Obr.: Rozdělení objektu pravidelnou mřížkou a výběr reprezentativních bodů

2.2.2.1.2 Konstrukce BSP

Každý reprezentativní bod si nese s sebou informaci v podobě Quadric error metric (QEM) a Dual quadric error metric (DQEM). Lze sestavit BSP strom pomocí následujících pravidel:

- kořen je bod určený sumou QEM všech reprezentativních bodů. Nese si s sebou také sumu DQEM
- další uzly se získají vytvořením dělící roviny pomocí DQEM rodičovského uzlu a rozdělením reprezentativních bodů připadajících rodiči podle této dělící roviny na dvě skupiny. Každá skupina vytváří opět bod, který je určen sumou QEM reprezentativních bodů do skupiny spadajících.
- dělí se vždy uzel s největší QEM.

Pokud je BSP strom tvořen tímto způsobem, je možné kdykoliv přestat a přejít k dalšímu kroku. Výsledky se budou lišit pouze úrovní detailu.

2.2.2.1.3 Zjednodušení

V poslední fázi algoritmu se prochází všechny trojúhelníky původní trojúhelníkové sítě a kontroluje se, zda body, které jej tvoří, spadají do třech různých listů BSP stromu. Pokud tuto podmínku nesplňují, jsou vyřazeny. Pokud ano, jsou spojeny body určené těmito listy BSP stromu do trojúhelníku. Takto se vytvoří trojúhelníková síť nad všemi listy BSP stromu.

2.2.2.1.4 Zhodnocení metody

Jde o globální metodu zjednodušování. Mohla by se jevit jako zajímavá s ohledem na lokalizaci zobrazení – díky uniformnímu dělení na počátku. Garland také dále v současné době rozšiřuje metodu a ukazuje sílu jejího použití v kombinaci s kontrakcí hrany pomocí QEM. Kombinací těchto metod lze dále urychlit proces zjednodušování.

2.3 Multitriangulace

V této práci nebudou zmiňovány metody zabývající se výstavbou hierarchie pomocí vloženého dělení (nested subdivision) , žádné metody zabývající se budováním hierarchie pravidelných sítí a také metody ukládání úrovní detailu. Budou zde ukázány jen metody pracující s obecnou, nepravidelnou trojúhelníkovou sítí.

2.3.1 Progresivní trojúhelníkové sítě [hopp9601],[hopp9801]

[progressive meshes]

Tento multiresolution model je založen na zjednodušování trojúhelníkové sítě pomocí „Kontrakce hran“ (viz. Zjednodušování). Při kontrakci hrany ubudou z původního objektu dvě plochy a jeden bod. Postupnou aplikací kontrakce tak získáváme posloupnost úrovní detailu od nejjemnějšího do nejhrubšího.

$$M^{orig} = M^{n} \xrightarrow{ecol_{n-1}} M^{n-1} \xrightarrow{ecol_{n-2}} \dots \xrightarrow{ecol_{1}} M^{1} \xrightarrow{ecol_{0}} M^{0}$$

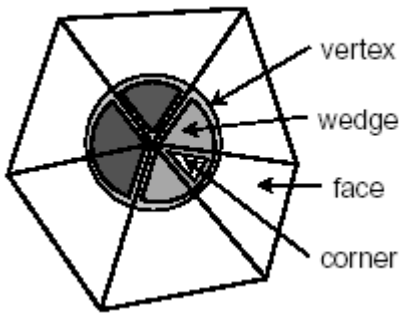
Ke kontrakci existuje inverzní operace rozštěpení bodu (vertex spliting). Aplikováním štěpení bodu na nejprve nejhrubší úroveň detailu získáme posloupnost opačnou k předchozí.

$$M^{0} \xrightarrow{vsplit_{0}} M^{1} \xrightarrow{vsplit_{1}} \dots \xrightarrow{vsplit_{n-2}} M^{n-1} \xrightarrow{vsplit_{n-1}} M^{n} = M^{orig}$$

Progresivní reprezentace trojúhelníkové sítě M je potom definována jako množina o dvou prvcích – počáteční nejhrubší trojúhelníková síť M^0 a množina všech operací štěpení bodu uložená sekvenčně.

$$PM = \{M^{0}, \{vsplit_{0}, vsplit_{1}, \dots, vsplit_{n-1}\}\}$$

Hoppe ukazuje [hopp9601] a [hopp9801], že je výhodné používat tzv. klínovou datovou reprezentaci trojúhelníkové sítě. Tato struktura vhodně popisuje různé nespojitosti povrchu (nespojité normály) a přidaných atributů (různé barvy sousedních trojúhelníků, změna textury, UV souřadnice textury , atd.).



Blíže je tato struktura v pseudo-c notaci popsána v [hopp9801]. Hoppe také zavádí tzv. Geomorf, jenž umožňuje plynule interpolovat přechod mezi dvěmi po sobě nenásledujícími úrovněmi detailu. Každý bod jemnějšího detailu je odvozen od právě jednoho rodičovského

bodu v hrubším detailu. Při znalosti souřadnic obou bodů je tedy možné mezi nimi vytvořit vizuálně plynulý přechod.

2.3.1.1 Zhodnocení metody

Metoda má mnoho kladů počínaje poměrně jednoduchou a průhlednou implementací, přes benevolenci při výběru ohodnocujícího algoritmu pro kontrakci hran a konče poměrně spolehlivým chováním, což byl zřejmě důvod, proč Microsoft zařadil tuto praktiku do DirectX 8.0.

2.3.2 Hierarchie bodů

[vertex hierarchy]
Tato metoda vychází z progresivních trojúhelníkových sítí a hlavním rozdílem je ukládání aplikovaných změn namísto sekvenčně do stromové struktury. Uzel stromu je adekvátní kontrakci hrany, kde rodiče tvoří bod vzniklý kontrakcí a potomky tvoří vrcholy kontrahované hrany. Cílem pro výstavbu každé úrovně stromu je provést co nejvíce nezávislých kontrakcí najednou. Po uzavření každé úrovně se propagují nepoužité vrcholy o úroveň výše.
Během extrakce se musí dodržovat určitá omezení, protože i přes negativní test na překlopení a zdvojení trojúhelníků je možné získat špatnou konfiguraci. Tato omezení jsou následující:

- vrchol v může být rozštěpen pouze v případě, že jsou rozbaleny všechny jeho okolní vrcholy
- hrana v může být kontrahována pouze tehdy, když všechny okolní vrcholy obou vrcholů hrany jsou již aktuální

Strom se tedy prochází dvojfázově. První fáze probíhá směrem zdola nahoru. Všechny vrcholy, které jsou pro dané zobrazení nevhodné, se vypustí na základě určitého LOD kriteria. Pak se ale kvůli výše uvedeným podmínkám musí kontrolovat přítomnost následníků těchto pozůstalých trojúhelníků.

2.3.2.1 Zhodnocení metody

Nevýhodou metody je způsob extrakce, kdy se pro získání trojúhelníkové sítě, třeba i nízkého rozlišení, musí procházet všechny trojúhelníky

2.3.3 Explicitní a implicitní multitriangulace

[explicit and implicit MT]
Algoritmus explicitní multitriangulace je postaven na struktuře orientovaného acyklického grafu(Directed acyclic graph). Uzly grafu tvoří fragmenty, které přímo uchovávají lokální seznam trojúhelníků před změnou a po změně. Hrany grafu vytvářejí závislosti fragmentů. Tato struktura může být jednoduše traverzovaná při průchodu směrem od kořene a v seznamu aktuálních trojúhelnlníků zůstanou jen trojúhelníky, které tvoří souvislou trojúhelníkovou síť v určitém detailu. Nad tímto seznamem se ještě provede vyřazení nepotřebných trojúhelníků.

Implicitní trojúhelníkové sítě na rozdíl od explicitních neuchovávají redundantní informaci o trojúhelnících fragmentu, ale uchovávají jen informaci o změně podobně jako progresivní t.s. nebo hierarchie bodů.

2.3.3.1 Zhodnocení metody

Explicitní trojúhelníkové sítě jsou velmi rychle zobrazitelné, jsou kdykoliv přerušitelné (při zachování atomičnosti operace štěpení bodu). To jsou vlastnosti, které tuto metodu předurčují

k použití v aplikacích reálného času. Nevýhodou je velká datová náročnost. Naopak implicitní t.s. snižují oproti explicitním t.s. datovou, ale zvyšují časovou náročnost.

2.4 Metody zvolené k implementaci a jejich bližší popis

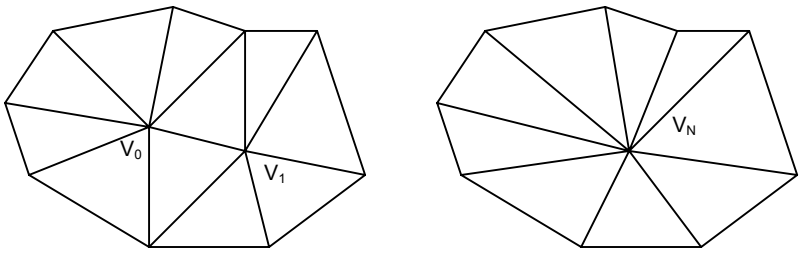
Jako zjednodušující metoda byla zvolena kontrakce hrany pomocí kvadrické chybové metriky. Tato metoda vykazuje vizuálně dobré výsledky, kontroluje globální chybu zjednodušované sítě i přes drobnou úlevu na přesnosti popsanou v této části. V dnešní době je hojně využívána jak při zjednodušování za účelem výstavby multitriangulace, tak za účelem jednorázového zjednodušení trojúhelníkové sítě. Její implementace je zařazena i mezi základní plug-in moduly 3D editoru Lightwave. Metoda je velice rychlá ve výstavbě. Dosahuje složitosti $O(N \log N)$ - vytvoření fronty a $O(N \log N)$ - zpracování fronty.

Metoda zvolená pro výstavbu multitriangulace je explicitní MT. Lákavá je zejména její vlastnost rychlé extrakce, jednoduchá implementace a dobré vlastnosti pro selektivní extrakce. Metoda sice vykazuje velkou paměťovou náročnost, která však přiměřeně kompenzuje současný standard velikosti operační paměti. Paměťová složitost zůstává stále $O(N)$, i když koeficient u N je poměrně veliký.

2.4.1 Zjednodušení pomocí kontrakce hrany

2.4.1.1 Základní kontrakce

Základem této metody je kontrakce hrany. Kontrakce hrany spočívá ve výběru potencionální dvojice bodů a jejich následném spojení do jediného bodu. Tento bod může být vybrán z množiny původních bodů nebo může jít o bod úplně nový s optimálním umístěním ve smyslu určitého kriteria.

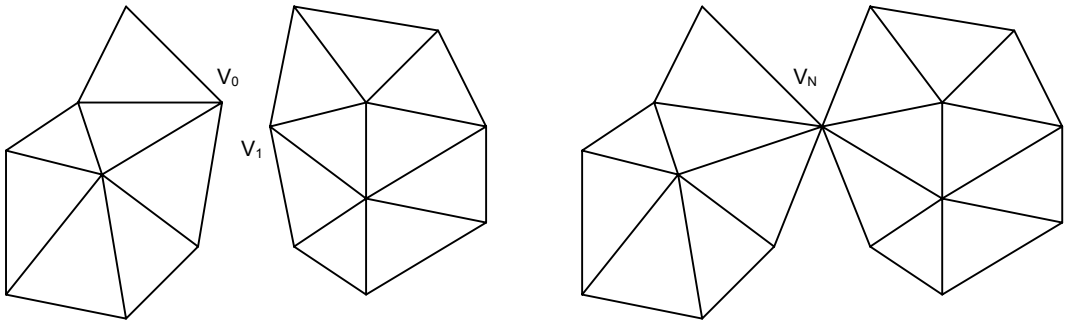


Obr.: Kontrakce hrany

Výhodou této zjednodušovací operace je i její aplikace na hrany nonmanifold trojúhelníkových sítí.

2.4.1.2 Kontrakce neexistující hrany

Při této operaci z původní trojúhelníkové sítě ubudou dva trojúhelníky, tři hrany a jeden bod. Existují také různé modifikace této operace na kontrakci neexistujících hran. Taková operace je schopná zjednodušovat i neúplné a rozdrobené objekty (např. pole čtyřstěnů, které by více již nešlo zjednodušit).



Obr.: Kontrakce neexistující hrany

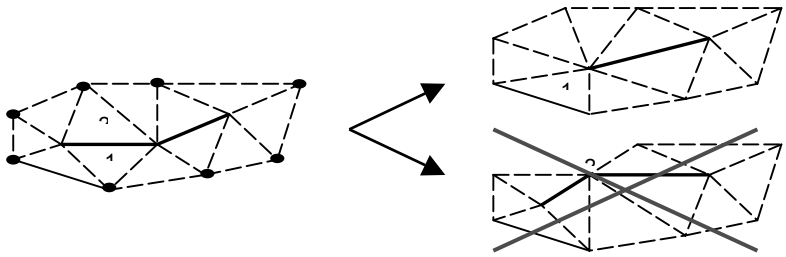
2.4.1.3 Iterativní kontrakce hran

2.4.1.3.1 Konvergence zjednodušení

Pokud provádíme operaci kontrakce hrany iterativně na trojúhelníkové síti s konečným počtem bodů, ubude v každém kroku jeden bod, z čehož je zřejmá konvergence k již dále nezjednodušitelné síti v konečném počtu kroků. Počet kroků je tedy shora omezen počtem vrcholů a zpravidla je o něco menší protože iterace je ukončena nějakou podmínkou uspokojivého výsledku.

2.4.1.3.2 Ochrana hranic objektů

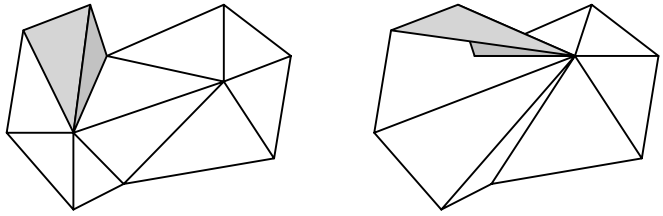
Někdy je při zjednodušování důležité zachovat hranici původního objektu. V tomto případě je nutné zakázat kontrakci hran, jejichž vrchol tvoří hranici objektu.



Obr.: Ochrana hranice objektů při kontrakci hran

2.4.1.3.3 Ochrana před přehnutím trojúhelníkové sítě

V určité konfiguraci trojúhelníků zainteresovaných do kontrakce hran může dojít k tzv. přehnutí trojúhelníkové sítě. Jde o jev, kdy normály dvou sousedních trojúhelníků svírají po kontrakci hrany úhel větší než $\frac{\pi}{2}$.



Obr.: Přehnutí trojúhelníkové sítě při kontrakci hran

2.4.2 Odvození kvadrické chybové metriky (Quadric error metric)

Mějme dán bod a rovinu v E^3 prostoru. Jeho kvadratická vzdálenost od roviny se dá vyjádřit jako:

$$\begin{aligned} D^2(\mathbf{v}) &= (\mathbf{n}^T\mathbf{v} + d)^2 \\ &= (\mathbf{v}^T\mathbf{n} + d)(\mathbf{n}^T\mathbf{v} + d) \\ &= \mathbf{v}^T\mathbf{n}\mathbf{n}^T\mathbf{v} + 2d\mathbf{n}^T\mathbf{v} + d^2 \\ &= \mathbf{v}^T(\mathbf{n}\mathbf{n}^T)\mathbf{v} + 2(d\mathbf{n})^T\mathbf{v} + d^2 \end{aligned}$$

Matice $\mathbf{n}\mathbf{n}^{\mathbf{T}}$ je pozitivně semidefinitní matice 3x3. Označme si ji jako $\mathbf{A}$. Matici 3x1 $d\mathbf{n}$ si označíme jako $\mathbf{b}$ a skalár d^2 si označíme jako c. Dostaneme tedy následující vztah:

$$D^2(\mathbf{v}) = \mathbf{v}^{\mathbf{T}}\mathbf{A}\mathbf{v} + 2\mathbf{b}^{\mathbf{T}}\mathbf{v} + c$$

Součet kvadratických vzdáleností bodu od množiny rovin lze vyjádřit jako:

$$\begin{aligned}\sum D^2(\mathbf{v}) &= \sum(\mathbf{v}^{\mathbf{T}}\mathbf{A}\mathbf{v} + 2\mathbf{b}^{\mathbf{T}}\mathbf{v} + c)\\ &= \sum \mathbf{v}^{\mathbf{T}}\mathbf{A}\mathbf{v} + \sum 2\mathbf{b}^{\mathbf{T}}\mathbf{v} + \sum c\\ &= \mathbf{v}^{\mathbf{T}}(\sum \mathbf{A})\mathbf{v} + 2(\sum \mathbf{b})^{\mathbf{T}}\mathbf{v} + (\sum c)\end{aligned}$$

Tento vztah nám říká, že pokud chceme získat součet vzdáleností bodu od množiny ploch, stačí pouze sečíst základní matice A,b,c všech rovin a pak je dosadit do rovnice kvadratické vzdálenosti.

Analogie kvadratické vzdálenosti od roviny a izoplochy kvadriky
Matice $\mathbf{A}$ je symetrická a lze tedy rozepsat následujícím způsobem:

$$\mathbf{A} = \begin{bmatrix} a & b & d \\ b & c & e \\ d & e & f \end{bmatrix}$$

matice $\mathbf{b}$ pak jako:

$$b = \begin{bmatrix} g \\ h \\ i \end{bmatrix}$$

a skalár c označíme jako j. Rovnici kvadratické vzdálenosti lze tedy rozepsat:

$$D^2(\mathbf{v}) = (v_x{}^2 a + v_y{}^2 c + v_z{}^2 f + 2v_x v_y b + 2v_x v_z d + 2v_y v_z e) + (2v_x g + 2v_y h + 2v_z i) + j$$

Uvažujme nyní kvadriku **Q**:

$$\mathbf{Q} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{12} & a_{22} & a_{23} & a_{24} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix}$$

Obecná rovnice izoplochy kvadriky **Q** má tvar (**v** je v homogenních souřadnicích):

$$e = \mathbf{v}^{\mathrm{T}}\mathbf{Q}\mathbf{v}$$

$$e = v_x{}^2 a_{11} + v_y{}^2 a_{22} + v_z{}^2 a_{33} + 2v_x v_y a_{12} + 2v_x v_z a_{13} + 2v_y v_z a_{23} + 2v_x a_{14} + 2v_y a_{24} + 2v_z a_{34} + a_{44}$$

Na první pohled je patrná analogie s kvadratickou vzdáleností bodu od roviny

$$\mathbf{Q} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{12} & a_{22} & a_{23} & a_{24} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix} = \begin{bmatrix} a & b & d & g \\ b & c & e & h \\ d & e & f & i \\ g & h & i & j \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^{\mathrm{T}} & c \end{bmatrix}$$

Protože kvadrická forma lze vyjádřit jako vážený součet tří nezávislých lineárních forem

$$A = a_1 L_1{}^2 + a_2 L_2{}^2 + a_3 L_3{}^2$$

a vzdálenost od roviny je lineární formou, určuje kvadratická vzdálenost od aspoň tří rovin kvadriku **Q**.

Submatice **A** je pozitivně semidefinitní (všechna vlastní čísla jsou nezáporná), tedy izoplochy kvadriky **Q** jsou elipsoidy, které mohou za určitých podmínek zdegenerovat (matice **A** je singulární). Z pohledu rovin mohou nastat tyto konfigurace degenerace elipsoidu:

- všechny roviny tvořící Q jsou paralelní s libovolnou přímkou → izoplochy jsou nekonečné válce s osou v přímce
- všechny roviny tvořící Q jsou rovnoběžné → izoplochy jsou rovnoběžné roviny

Hledání minimální vzdálenosti
Postačující aproximací minimální vzdálenosti od množiny rovin bude pro nás minimální Q(v). Parciální derivace musí být nulové tedy $\frac{\partial Q(\mathbf{v})}{\partial x} = \frac{\partial Q(\mathbf{v})}{\partial y} = \frac{\partial Q(\mathbf{v})}{\partial z} = 0$. Vyjdeme-li z rozepsané rovnice kvadratické vzdálenosti dostaneme tyto vztahy:

$$\left.\begin{aligned} \frac{\partial Q(\mathbf{v})}{\partial x} &= 2av_x + 2bv_y + 2dv_z + 2g \\ \frac{\partial Q(\mathbf{v})}{\partial y} &= 2bv_x + 2cv_y + 2ev_z + 2h \\ \frac{\partial Q(\mathbf{v})}{\partial z} &= 2dv_x + 2ev_y + 2fv_z + 2i \end{aligned}\right\} \nabla Q(\mathbf{v}) = 2\mathbf{A}\mathbf{v} + 2\mathbf{b}$$

minimum tedy nastane v bodu **v**, který lze snadno určit

$$\begin{aligned} \nabla Q(\mathbf{v}) &= \mathbf{o} \\ 2\mathbf{A}\mathbf{v} + 2\mathbf{b} &= \mathbf{o} \\ \overline{\mathbf{v}} &= -\mathbf{A}^{-1}\mathbf{b} \end{aligned}$$

Pokud je matice **A** singulární, tedy neinvertovatelná, tvoří množina bodů s minimální kvadratickou vzdáleností přímku nebo rovinu. Dosadíme-li zpětně dostaneme i hodnotu nejmenší vzdálenosti:

$$\begin{aligned} Q(\mathbf{v}) &= \mathbf{v}^{\mathbf{T}}\mathbf{A}\mathbf{v} + 2\mathbf{b}^{\mathbf{T}}\mathbf{v} + c \\ Q(\overline{\mathbf{v}}) &= (-\mathbf{A}^{-1}\mathbf{b})^{\mathbf{T}}\mathbf{A}(-\mathbf{A}^{-1}\mathbf{b}) + 2\mathbf{b}^{\mathbf{T}}(-\mathbf{A}^{-1}\mathbf{b}) + c \\ Q(\overline{\mathbf{v}}) &= \mathbf{b}^{\mathbf{T}}(\mathbf{A}^{-1})^{\mathbf{T}}\mathbf{b} - 2\mathbf{b}^{\mathbf{T}}\mathbf{A}^{-1}\mathbf{b} + c \\ Q(\overline{\mathbf{v}}) &= \mathbf{b}^{\mathbf{T}}\mathbf{A}^{-1}\mathbf{b} - 2\mathbf{b}^{\mathbf{T}}\mathbf{A}^{-1}\mathbf{b} + c \\ Q(\overline{\mathbf{v}}) &= -\mathbf{b}^{\mathbf{T}}\mathbf{A}^{-1}\mathbf{b} + c \end{aligned}$$

2.4.2.1 Shrnutí důležitých vlastností

Každá rovina definuje symetrickou pozitivně semidefinitní chybovou matici **Q**

$$\mathbf{Q} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^{\mathbf{T}} & c \end{bmatrix} \text{, kde}$$

$$\mathbf{A} = \mathbf{n}\mathbf{n}^{\mathbf{T}},\ \mathbf{b} = d\mathbf{n},\ c = d^2$$

Kvadratickou vzdálenost bodu **v** od roviny lze vypočítat jako

$$Q(\mathbf{v}) = \mathbf{v}^{\mathbf{T}}\mathbf{A}\mathbf{v} + 2\mathbf{b}^{\mathbf{T}}\mathbf{v} + c$$

Výpočet kvadratické vzdálenost bodu **v** od množiny rovin je adekvátní výpočtu kvadratické vzdálenosti bodu od jedné „roviny“ s chybovou maticí rovnou součtu chybových matic všech rovin, tedy formálně:

$$\sum Q(\mathbf{v}) = (\sum Q)(\mathbf{v})$$

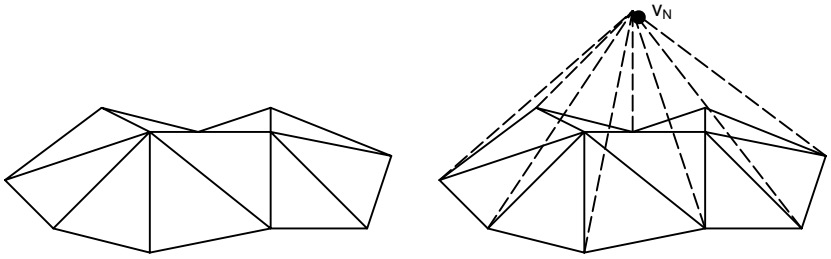
Nalezení místa s nejmenší kvadratickou vzdáleností od množiny aspoň tří rovin a výpočet její velikosti je adekvátní nalezení středu (možná degenerovaného) elipsoidu určeného součtem jejich chybových matic a jeho zpětného dosazení do rovnice kvadratické vzdálenosti.

$$\overline{\mathbf{v}} = -\mathbf{A}^{-1}\mathbf{b}$$

$$Q(\overline{\mathbf{v}}) = -\mathbf{b}^{\mathbf{T}}\mathbf{A}^{-1}\mathbf{b} + c$$

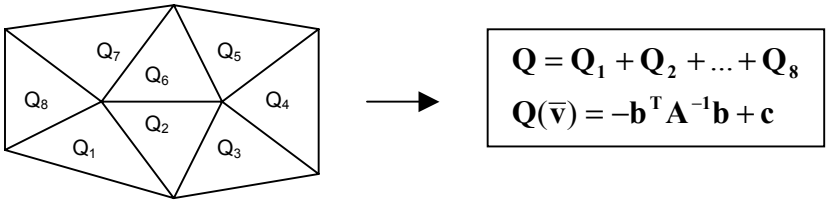
2.4.2.2 Využití Q při ohodnocování hran ke kontrakci

Při kontrakci hrany vzniká nový bod nebo se použije jeden z bodů původně tvořících hranu. Jednou z možností, jak ohodnotit kontrakci hrany je výpočet vzdálenosti(kvadratické) tohoto bodu od všech rovin, které incidují s oběma vrcholy. Pokud se tedy pro každou hranu určí poloha bodu vzniklého její kontrakcí, je možné tyto kontrakce ohodnotit pomocí kvadratické vzdálenosti odvozené v předchozí kapitole.



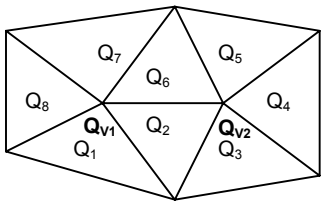
Obr.: Umístění nového bodu pro ohodnocení kontrakce

Každý trojúhelník této „záplaty“ určuje svou chybovou matici Q. Za předpokladu, že tuto „záplatu“ tvoří alespoň tři trojúhelníky, lze součtem chybových matic získat qadriku **Q** a nalézt její střed – tedy bod, který má nejmenší kvadratickou vzdálenost ke všem přilehlým rovinám (trojúhelníkům). V případě, že tento bod nelze určit (**A** je singulární), zvolí se jinou metodou např. vybráním jednoho z původních nebo zvolením bodu v polovině hrany.



Obr.: Ohodnocení hrany ke kontrakci pomocí chybových matic rovin

Takto by se ovšem ohodnocovaly všechny hrany složitě. Pro všechny hrany by bylo nutné sečíst všechny přilehlé Q v době jejich vyhodnocování. Garland ukázal, že dostatečnou aproximací ohodnocení hrany je výpočet zatížený chybou duplikování některých trojúhelníků. Za tohoto předpokladu je možné udržovat si kvadriku určenou rovinami přilehlých trojúhelníků v každém vrcholu trojúhelníkové sítě. Při vyhodnocování hrany se tedy pouze sečtou kvadriky obou vrcholů.

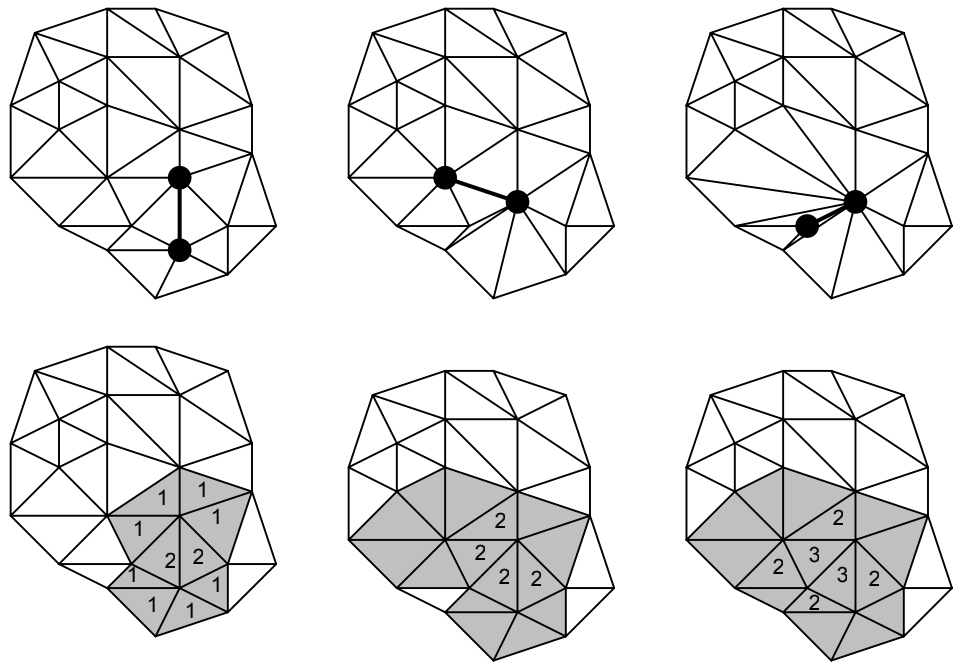


$$\mathbf{Q}_{V1} = \mathbf{Q_1} + \mathbf{Q_2} + \mathbf{Q}_5 + \mathbf{Q_7} + \mathbf{Q_8}$$
$$\mathbf{Q}_{V2} = \mathbf{Q}_2 + \mathbf{Q}_3 + \mathbf{Q}_4 + \mathbf{Q}_5 + \mathbf{Q}_6$$
$$\mathbf{Q} = \mathbf{Q_{V1}} + \mathbf{Q_{V2}} - \varepsilon = \mathbf{Q_A} - \varepsilon$$
$$\varepsilon = \mathbf{Q_2} + \mathbf{Q_6}$$
$$\mathbf{Q}(\overline{\mathbf{v}}) \doteq \mathbf{Q_A}(\overline{\mathbf{v}}_A)$$

Obr.: Ohodnocení hrany zatížené chybou

2.4.2.3 Využití Q při kontrakci hrany

Aplikuje-li se kontrakce hrany iterativně v trojúhelníkové síti, kvadriky nových bodů jsou tvořeny ze stále větších „záplat“ původní trojúhelníkové sítě. Jeden trojúhelník se může vyskytnout v ohodnocení určité hrany maximálně třikrát. Toto násobné počítání lze eliminovat určitými technikami, avšak Garland ve své práce uvádí, že nemá na výslednou aproximaci významný vliv. Proto je výhodné tento fakt v aplikaci zanedbat.



Obr.: Trojúhelníky původní trojúhelníkové sítě zainteresované do ohodnocení hrany a jejich četnost

Každý nový bod má přiřazenu matici **Q** tvořenou součtem matic $\mathbf{Q_1}$ a $\mathbf{Q_2}$ koncových bodů hrany, jejíž kontrakcí bod vznikl. Jeho poloha je určena minimalizací $\mathbf{Q}(\mathbf{v})$. V případě, že nelze minimum určit, zvolí se poloha bodu jinou strategií. Je zřejmé, že s přibývajícím počtem trojúhelníků v “záplatě“ klesá pravděpodobnost vzniku singularity (pokud nejde o rozsáhlou rovinu tvořenou velkým počtem trojúhelníků nebo trojúhelníky na válcové ploše **viz** vznik singularit).

2.4.3 Popis DAG struktury (hierarchie kauzalit)

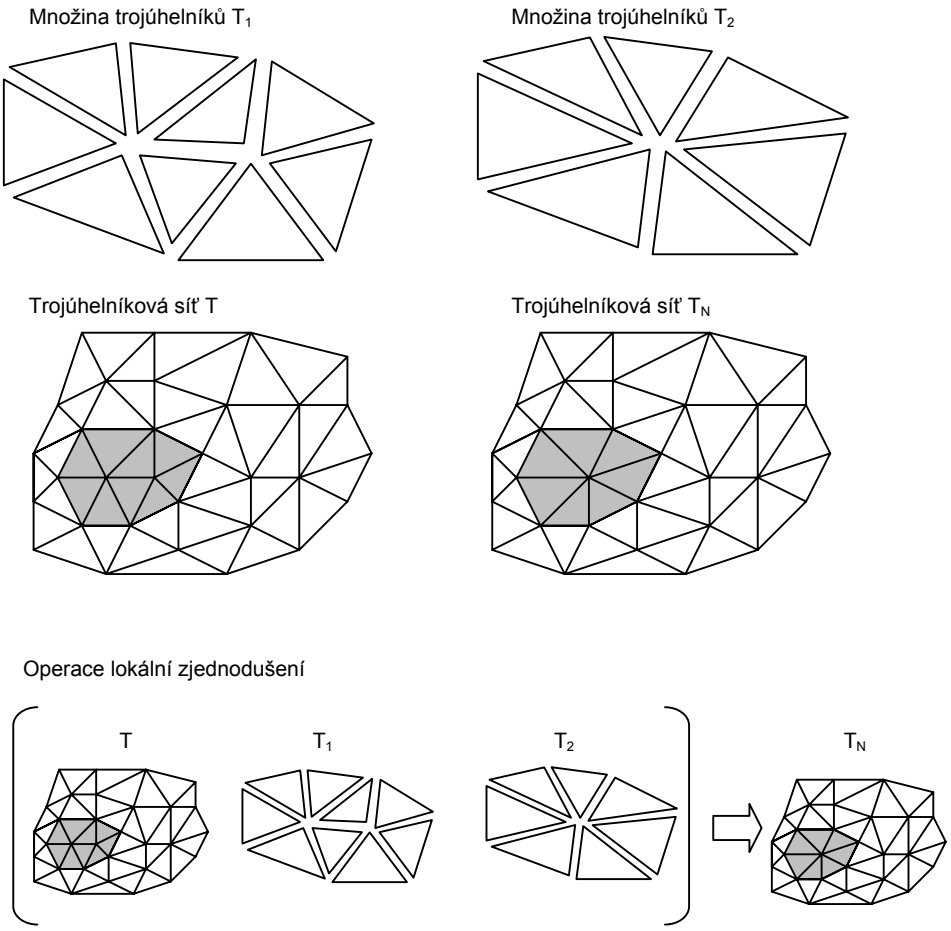
Podíváme-li se na postup provádění lokálních změn v trojúhelníkové síti při zjednodušování pomocí kontrakce hrany, můžeme definovat následující pojmy:

Operace ***lokální změna*** $L(T,T_1,T_2)=T_N$, kde $T_1 \subset T \wedge T_2 \subset T$ nad trojúhelníkovou sítí T je složená z vypuštění vstupních trojúhelníků T1 z trojúhelníkové sítě T a vložení trojúhelníků T2 do trojúhelníkové sítě. Pokud $T_1 \neq T_2$ vzniká nová trojúhelníková síť. Tato operace je invertibilní.

Operace lokální změny $L(T,T_1,T_2)=T_N$,kdy $|T_1|>|T_2|$ se nazývá ***lokální zjednodušení*** a operace $L(T,T_1,T_2)=T_N$, kdy $|T_2|>|T_1|$ se nazývá ***lokální zjemnění***.

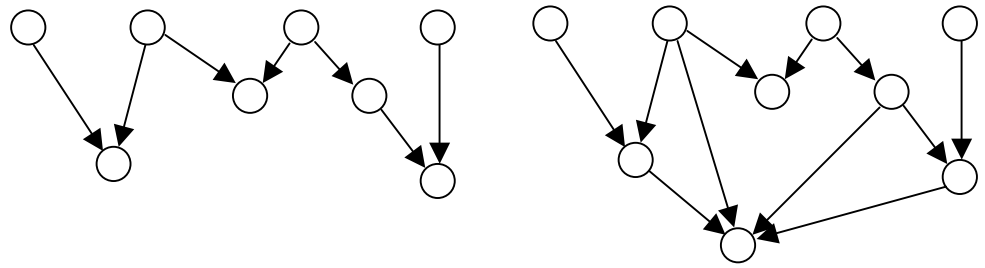
Mějme dvě lokální změny L1, L2. Má-li množiny T2 lokální změny L1 neprázdný průnik s množinou T1 lokální změny L2, říkáme, že L2 je ***přímo závislá*** na L1.

Z pohledu zjednodušování pomocí kontrakce hrany vypadají množiny operace lokální zjednodušení T1 a T2 takto (při lokálním zjemnění se obrátí):
T1 - množina všech trojúhelníků incidujících aspoň s jedním vrcholem hrany
T2 – množina všech trojúhelníků incidujících s nově vyniklým bodem



Obr.: Schématické znázornění operace zjednodušení

Na počáteční trojúhelníkovou síť jsou během zjednodušování postupně aplikovány lokální změny. Zaznamená-li se během procesu redukce přímá závislost mezi jednotlivými změnami do grafu, kde uzly tvoří lokální změny a orientované hrany přímé závislosti s ohledem na zjednodušování, vznikne tak orientovaný acyklický graf (directed acyclic graph – DAG). Tento graf má pouze jeden koncový uzel, který reprezentuje nejjednodušší trojúhelníkovou síť.



Obr.: Postupné vytváření orientovaného acyklického grafu během procesu zjednodušování

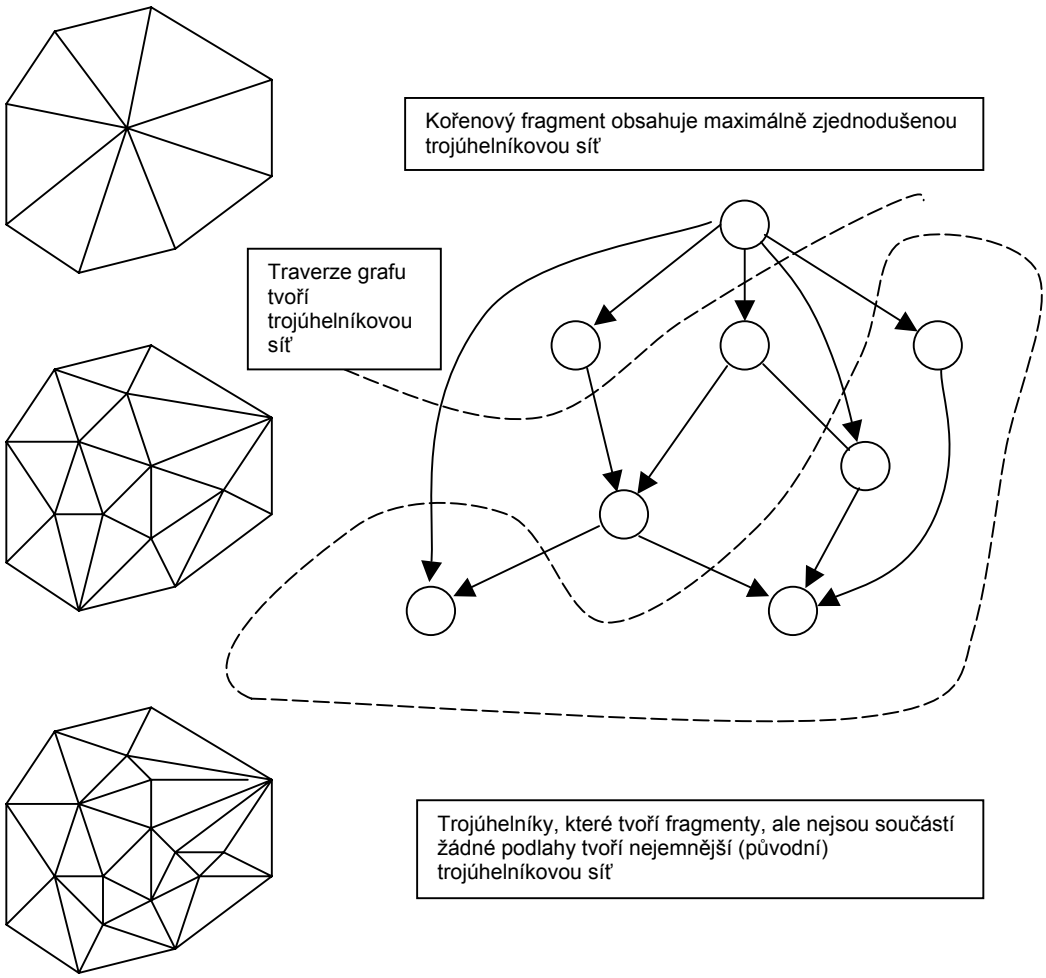
Pro potřeby extrakce trojúhelníkové sítě směrem od nejjednoduššího modelu k složitějšímu není takto orientovaný graf příliš vhodný a je tedy nutné všechny jeho hrany přeorientovat. Výsledná podoba grafu je tedy na světě. Tento graf má některé zajímavé vlastnosti.

Ad 1)
Nejjednodušší trojúhelníková síť je uložena v jeho kořeni, který jako jediný uzel popisuje změnu s prázdnou vstupní množinou trojúhelníků.

Ad 2)
Koncové uzly grafu vytváří část trojúhelníků nejjemnějšího rozlišení. Zbytek je uložen mezi ostatními uzly, jejichž některé trojúhelníky byly použity u jejich následníků.

Ad 3)
Traverzí grafu je možné získat trojúhelníkovou síť s počtem trojúhelníků ležícím mezi počtem trojúhelníků nejjemnějšího a nejjednoduššího rozlišení.

Ad 4)
Graf není nutné procházet přesně v opačném pořadí změn prováděných při jeho sestavování. Některé změny jsou totiž vzájemně nezávislé.



Obr.: Význam grafu pro získání trojúhelníkové sítě určité kvality (rozlišení)

2.4.3.1 Struktura fragmentu

Fragment reprezentuje lokální změnu trojúhelníkové sítě, tedy nese veškeré informace o provedené změně, její lokalitě a jejím umístění v hierarchii všech provedených změn. Formálně lze tedy popsat fragment jako:

$$F = \{S, L, H\}, \text{ kde}$$

S je množina atributů popisující změnu, L množina atributů lokalizace v trojúhelníkové síti a H je množina atributů popisující umístění změny vzhledem k hierarchii změn. Pokud se nepočítá s lokalizovanou extrakcí (podmíněnou extrakcí), není třeba uchovávat informace o lokalizaci změny a fragment se tím redukuje na

$$F = \{S, H\}$$

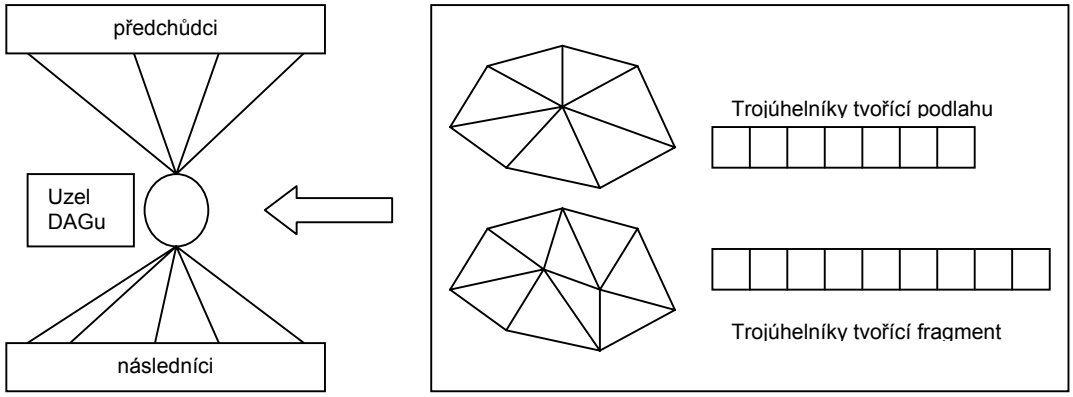
Takto definovaný fragment lze také použít, pokud sice uvažujeme lokalizovanou extrakci, ale pouze na bázi polohy vrcholů extrahovaných trojúhelníků.
V této práci budeme dále požívat neredukovanou formu fragmentu a jednotlivé množiny budou obsahovat tyto atributy:

S – trojúhelníky formující simplex, trojúhelníky vzniklé při kontrakci hrany (podlaha)
L – bod vzniklý kontrakcí hrany (reprezentant polohy změny)
H – reference na fragmenty závislé na změně reprezentované tímto fragmentem, reference na fragmenty vytvářející trojúhelníky podlahy

Struktura fragmentu zapsaná v jazyce C vypadá takto:

```
Struct Fragment
{
      int         pocetHornichReferenci;
      int         pocetDolnichReferenci;
      fRef        *refHorniFragmenty;
      fRef        *refDolniFragmenty;
      int         pocetTrojuhelnikuPodlahy;
      iTri        *indexyTrojuhelniku;
      iTri        *indexyTrojuhelnikuPodlahy;
      iBod        signifikantniBod;
}
```

Je zřejmé, že pro trojúhelníky a body jsou použité pouze indexy do pole všech bodů a do pole všech trojúhelníků. Z definice změny provedené kontrakcí hrany také vyplývá, že počet trojúhelníků podlahy odpovídá počtu trojúhelníků fragmentu zmenšený o dva zrušené trojúhelníky. Není proto třeba si jej navíc uchovávat. Počet horních a dolních referencí většinou neodpovídá počtu trojúhelníků podlahy a počtu trojúhelníků fragmentu, protože jen zřídka všechny následující změny využijí jen jeden trojúhelník tohoto fragmentu.



Obr.: Struktura fragmentu orientovaného acyklického grafu (DAG)

2.4.3.2 Vytvoření grafu během zjednodušování

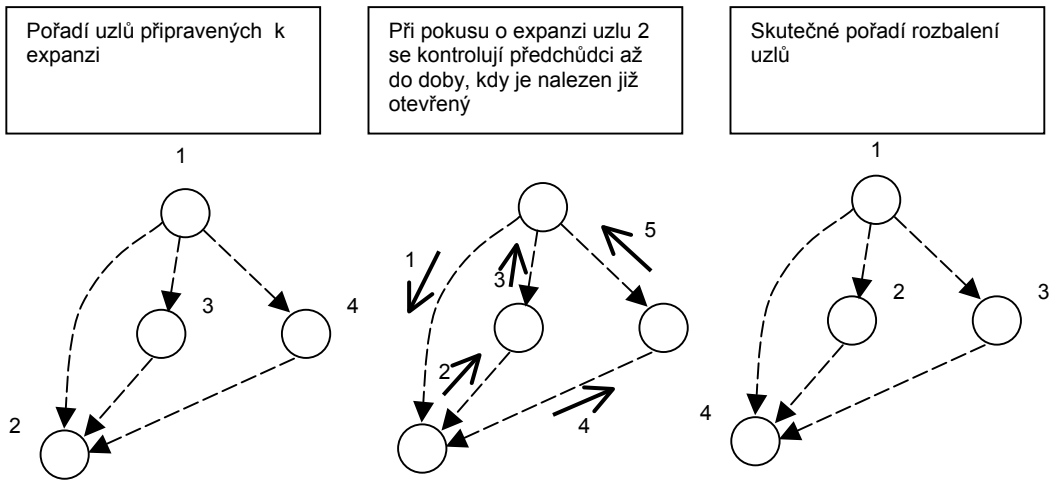
Acyklický orientovaný graf je stavěn směrem od listů ke kořeni. Při inicializaci výstavby je sestrojeno pole ***TFConnection*** reprezentující spojení mezi trojúhelníky a již vytvořenými fragmenty. Pole je zpočátku naplněno samými hodnotami NULL. Pokud ještě není trojúhelník v žádném fragmentu, je jeho odkaz na fragment ohodnocen jako NULL. Vložení nového fragmentu probíhá podle následujícího pseudokódu:

```
VložFragment(nové trojúhelníky,trojúhelníky podlahy)
{
      Alokuj pole horních odkazů
      Spočti a alokuj spodní odkazy
      Naplň spodní odkazy podle TFConnection pole
      Pozměň horní odkazy následníků
      Pozměň odkazy na trojúhelníky podlahy v TFConnection poli
}
```

Protože dopředu není jasné, kolik předchůdců bude daný fragment mít, alokuje se stejně odkazů, jako je trojúhelníků v podlaze. Aby bylo možné později uvolnit přebytečné odkazy, má každý fragment přiřazeno počítadlo použitých trojúhelníků podlahy. V okamžiku, kdy je dosaženo celkového počtu, uvolní se přebytečné. Spodní odkazy se jednoduše zjistí pomocí TFConnection pole a alokace tedy proběhne na míru.

2.4.3.3 Extrakce trojúhelníkové sítě z grafu

Při pohybu orientovaným acyklickým grafem je třeba držet se určitých pravidel, aby byla zachována konzistence trojúhelníkové sítě. Jednotlivé uzly je možné rozbalovat teprve v případě, že všechny jeho předchůdci jsou již rozbaleny. Pokud by se tato podmínka neuplatňovala, vznikaly by v trojúhelníkové síti různé překryvy a díry.



Obr.: Kontrola rozbalených předchůdců zpětným procházením

V hierarchické struktuře typu DAG je možná celá škála různých strategií pohybu. Při extrakce hrajou důležitou roli čtyři faktory:

- zastavovací podmínka
- strategie pohybu v orientovaném acyklickém stromu (do šířky, do hloubky)
- inklusivně-exkluzivní pravidlo výběru trojúhelníků (fragmentů) podle uživatelem definované booleovské funkce
- heuristická pravidla pro řazení potencionálních uzlů grafu pro expanzi

V nejjednodušším případě se použije jen vybraná strategie pohybu v DAGu. Výsledkem je kompletně rozbalená trojúhelníková síť.

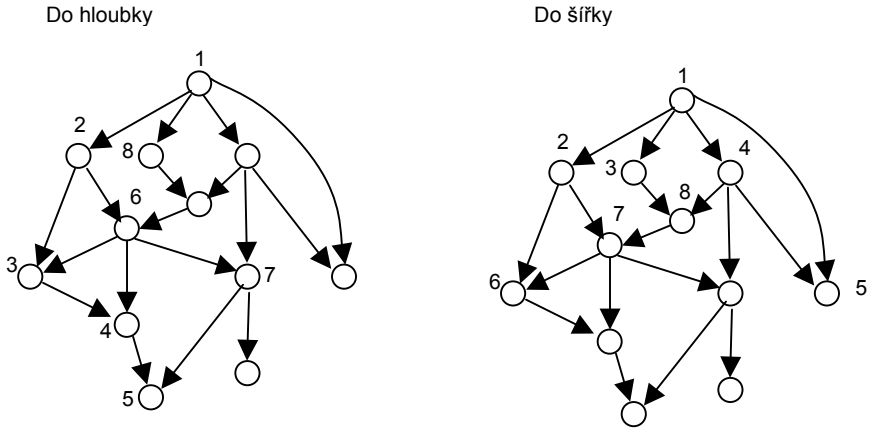
2.4.3.3.1 Zastavovací podmínka

Závisí na typu aplikace. Nejčastěji se jedná o dosažení určitého počtu trojúhelníků. Používá se v případě, kdy je důležité rychlé zobrazení s ohledem na možnosti grafického engine. To je hlavně u animací reálného času. Vzhledem k nárůstu počtu trojúhelníků s každým rozbaleným uzlem o dva, je možné požadovaný počet trojúhelníků dosáhnout víceméně přesně.
Další možnou zastavovací podmínkou je dosažení předem daného času na rozbalování. Aplikovat tuto zastavovací podmínku lze zejména u zobrazovacích programů. Ještě lépe je kombinovat ji s předchozí podmínkou a v případě, že uživatel nijak nezasahuje (rotace, posun atd.), je postupně zvětšován interval povoleného rozbalování. V okamžiku, kdy uživatel opět

projeví zájem zasahovat do průběhu zobrazování, přepne se na první podmínku, aby se udržela interaktivita.

2.4.3.3.2 Strategie pohybu

V zásadě je možné pohybovat se v grafu směrem do hloubky a do šířky.



Obr.: Pořadí uzlů při expanzi do šířky a do hloubky

Každá strategie má své opodstatnění a záleží opět na cíli, který je na aplikaci kladen. Prohledávání do hloubky má svůj význam zejména při získávání úzce lokalizovaného místa v kombinaci s inklusivně exkluzivní booleovskou funkcí. A je realizován přibližně tímto algoritmem:

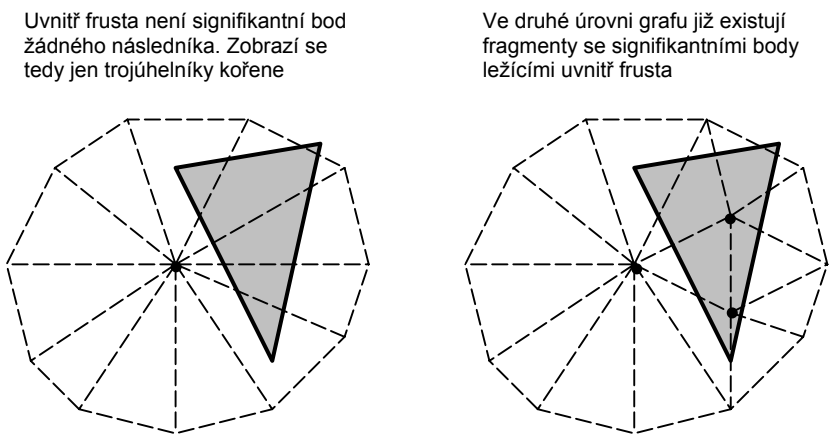
```
RozbalNásledníky(uzel)
{
      Otevři uzel
      Pro ∀následníka
            RozbalNásledníky(následník)
}

Rozbal()
{
      RozbalNásledníky(kořen)
}
```

Prohledávání do šířky má zase význam při rovnoměrné extrakci celého objektu při udržování rovnoměrného rozlišení po celé trojúhelníkové síti. Lze zapsat takto:

```
RozbalNásledníky(uzel,fronta)
{
      Otevři uzel
      Zařaď všechny následníky na konec fronty
}

Rozbal()
{
      Vlož do fronty kořen
      Dokud je ve frontě uzel
      {
            Vyber uzel ze začátku fronty
            RozbalNásledníky(uzel,fronta)
      }
}
```

2.4.3.3.3 Inkluzně-exkluzní pravidlo výběru trojúhelníků/fragmentů

V aplikaci může být potřeba vyřadit některé části trojúhelníkové sítě. To je možné dvěma způsoby. Nerozbalováním nepotřebných fragmentů nebo kompletním rozbalením, ale zavrhnutím nevyhovujících trojúhelníků. Vše se většinou provádí za pomocí booleovské funkce. První metoda vede k redukci procházené části grafu. Cenou za to jsou situace, kdy booleovská funkce zavrhne fragmenty, které by se měli zpracovávat. Příkladem je tomu případ, kdy se kamerou prochází nad povrchem a rozbalí se pouze ty fragmenty, jejichž signifikantní bod leží ve frustu kamery. Pokud je kamera příliš blízko povrchu a trojúhelníky kořene jsou příliš velké, hrozí, že nebude rozbalen žádný fragment a ačkoli uživatel bude očekávat detailní vykreslení, získá jen nejhrubší trojúhelníkovou síť. Řešením tohoto problému je vytvoření heuristického vkládání do fronty.



Obr.: Možnost nesprávného zobrazení detailu při vylučování nevyhovujících fragmentů

2.4.3.3.4 Heuristická pravidla pro řazení uzlů do fronty

Toto je nejsofistikovanější faktor při rozbalování trojúhelníkové sítě. Umožňuje rozbalování trojúhelníků přesně podle potřeby uživatele. Je možné naprogramovat heuristiku, která rozbaluje trojúhelníky tak, aby se při projekci na kameru jevily vzdálené i blízké trojúhelníky přibližně stejně veliké. Základem algoritmu je ohodnocení fragmentů čekajících na rozbalení a jejich následné řazení do fronty podle tohoto prioritního klíče. Nabízí se velké množství prioritních strategií.
Zde je uvedena jedna z nich:

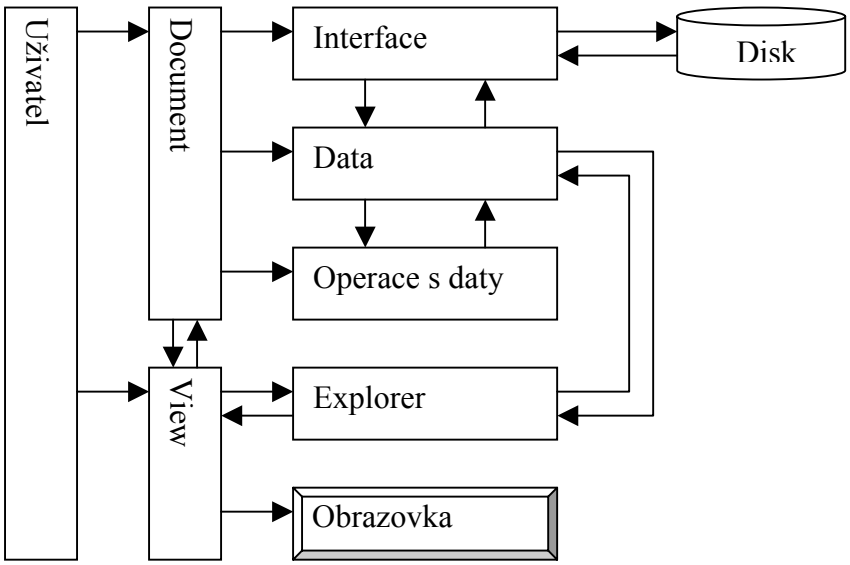
$$H = w_1 \cdot d_V + w_2 \cdot d_C \text{ , kde}$$

d_v je vzdálenost signifikantního bodu od směrového vektoru kamery a d_C je vzdálenost signifikantního bodu od kamery. Ohodnocení je rovno jejich váženému součtu. Je také nutné penalizovat fragmenty ležící za kamerou. Výhodné je tuto metodu kombinovat s inklusivně-exklusivní booleovskou funkcí a udržovat si dvě fronty fragmentů. Frontu fragmentů, které vyhovují booleovské funkci a frontu fragmentů nevyhovujících. Fronta vyhovujících je řazena pouze podle vzdálenosti od kamery. Fronta nevyhovujících je řazena podle kompletní formule. Prioritně se zpracovává fronta vyhovujících, a až teprve v okamžiku, kdy v ní už žádný fragment není, začne se zpracovávat fronta nevyhovujících.

```
RozbalNásledníky(uzel,fronta vyh., fronta nevyh.)
{
      Otevři uzel
      Pro ∀ následníka
            Vyhovuje boolovké funkci
            {
                  Ohodnoť podle vzdálenosti od kamery
                  Zařaď do fronty vyhovujících
            }
            Jinak
            {
                  Ohodnoť podle kompletní formule
                  Zařaď do fronty nevyhovujících
            }
}

Rozbal()
{
      Dokud není konec
      {
            Dokud není fronta vyh. prázdná
            {
                  Vyber uzel z fronty vyh.
                  RozbalNásledníky(uzel, fronta vyh.,
                                   fronta nevyh.)
            }
            Není konec
            {
                  Vyber uzel z fronty nevyh.
                  RozbalNásledníky(uzel, fronta vyh.,
                                   fronta nevyh.)
            }
      }
}
```

3 Realizační část

Aplikace je koncipovaná jako prohlížeč obecných dat vykreslených pomocí OpenGL. Navržena je tak, aby šlo pohodlně přidávat nové typy dat, aniž by se nějak zasahovalo do struktury programu. Vše je potom zakomponováno do struktury SDI (jednoduchý dokument / pohled model). Následující schéma znázorňuje funkční části aplikace.



Obr.: Funkční demodularizace aplikace

3.1 Funkční rozdělení aplikace

3.1.1 Vykreslování uživatelských dat

Explorer je navržen tak, aby nemusel vědět o vykreslovaných datech vůbec nic, kromě „bounding boxu“, který používá při umístění dat do základní polohy. Veškeré vykreslení si tak musí uživatel naprogramovat sám ve své třídě odvozené od základní CGLData. Ta definuje virtuální funkci GetBoundingBox a GLRender a GLPreRender. Při volání metod GLRender a GLPreRender je explorerem předávána datům informace o poloze a rotaci kamery. Metoda GLPreRender slouží k iniciování dat (například pro vykreslení do display listu).

3.1.2 Konverze dat

Je možné vytvořit konstrukci převádění dat jednoho typu na jiný. K tomu slouží další virtuální metody datového objektu LendData, ExportData, ImportData. Metoda LendData půjčí vnitřní data datového objektu jinému objektu, ale data si ponechá ve vlastnictví. Objekt půjčující data je tedy i nadále zodpovědný za jejich zrušení. Naproti tomu pomocí ExportData se data předávají dalšímu objektu i s vlastnictvím. Nový objekt je tedy zodpovědný za jejich zrušení. Metoda ImportData zase umožňuje data z jiného objektu přijímat. Konvertující objekt získá v konstruktoru oba datové objekty a v metodě ConvertData si podle potřeby vypůjčí nebo vezme vnitřní data z jednoho datového objektu, provede veškeré konvertující operace a vloží získaná data do druhého datového objektu.

3.1.3 Libovolná operace nad daty

Podobně jako při konverzi je tedy možné si od datového objektu vypůjčit nebo od něj převzít data. Tato data v nějakém funkčním objektu podle potřeby upravit a vše zase předat zpět původnímu objektu.

3.1.4 Načítání/ukládání dat do/ze souboru

Načítání není součástí uživatelsky definovaného objektu odvozeného od třídy CGLData. Namísto toho se využívají objekty rozhraní podle druhu souboru. Do konstruktoru je jim předáván objekt dat, který se má naplnit. Interface si zjistí o jaký typ dat jde a podle toho po zavolání metody ReadData načte soubor a zpracovaná data předá pomocí metody ImportData do objektu data. To umožňuje pro každý typ souboru vytvořit rozhraní, které je schopno komunikovat s různými typy dat. Stejně funguje i ukládání dat na disk pomocí metody WriteData, kde je voláno LendData nebo ExportData.

3.2 Datové objekty k dispozici

Jak již bylo zmíněno všechny datové objekty jsou odvozeny od společného objektu CGLData. V tomto odstavci budou popsány datové objekty, které jsou zatím k dispozici. Tyto objekty používají následující společné datové typy:

```
TVertex
{
      float x;
      float y;
      float x;
}

TTriangle
{
      int v1;
      int v2;
      int v3;
}
```

Výpis kódu: Společné datové typy

3.2.1 Základní třída datového objektu

```
class CGLData
{
protected:
      TBBox m_bBox;
      BOOL  m_bBoxFlag;

      virtual void generateBBox() = 0;
public:
      CGLData();
      virtual ~CGLData();

              TBBox   GetBBox();
      virtual int     GetDataType() = 0;
      virtual void    ImportData(void *i_pData) = 0;
      virtual void    LendData(void *o_pData) = 0;
      virtual void    ExportData(void *o_pData) = 0;
      virtual void    GLPreRender(TCMProps *i_pCmProps = 0) = 0;
      virtual void    GLRender(TCMProps *i_pCmProps = 0) = 0;
}
```

Výpis kódu: Základní třída datového objektu

3.2.2 Trojúhelníková síť definovaná třídou CGLTriData

Jde o jednoduchou trojúhelníkovou síť složenou z bodů a trojúhelníků s možností uchovat informace o normálách v bodech. Body jsou uloženy v poli struktur TVertex a trojúhelníky v poli struktur TTriangle. Takto vypadá deklarace třídy bez přetížených funkcí základní třídy.

```
class CGLTriData : public CGLData
{
private:
     BOOL       m_dataActive;

     GLuint     m_nTriangles;
     GLuint     m_nVertices;
     TVertex    *m_vertices;
     TVertex    *m_normals;
     TTriangle  *m_triangles;
}
```

Výpis kódu: Deklarace datového objektu jednoduché trojúhelníkové sítě bez přetěžovaných funkcí

Data jsou vykreslována předáním reference na pole bodů a reference na pole trojúhelníků serveru OpenGL. Není tedy využívána žádná urychlovací technika (pruhy trojúhelníků, vějíře trojúhelníků).

3.2.3 Hierarchická trojúhelníková síť definovaná třídou CGLMtrData

Tato třída obsahuje stejně jako třída CGLTriData pole trojúhelníků a pole bodů. Navíc je zde členský objekt typu CHierarchy. Tento objekt poskytuje metody pro vytváření hierarchie a pro extrakci trojúhelníkové sítě. Detailně bude pospán v další části.

```
class CGLMtrData : public CGLData
{
protected:
     GLuint     m_nTriangles;
     GLuint     m_nVertices;
     TVertex    *m_vertices;
     TVertex    *m_normals;
     TTriangle  *m_triangles;

     CHierarchy *m_pHierarchy;
}
```

Výpis kódu: Deklarace datového objektu hierarchické trojúhelníkové sítě

Při vykreslování je volána metoda Extract členského objektu CHierarchy. Po jejím provedení je zavolána metoda GetBuffer, která vrací ukazatel na pole trojúhelníků a samotné vykreslení proběhne opět stejně jako u jednoduché trojúhelníkové sítě. Objektu CHierarchy se při volání

metody Extract předává také ukazatel na booleovskou funkci a ukazatel na její parametry. Tato funkce je používána při rozbalování fragmentů hierarchie. Opět bude zmíněna v další části.

```
void CGLMtrData::GLRender(TCMProps *i_pCmProps)
{
      glColor3f(0.8f, 0.2f, 0.0f);

      TTriangle       *tmpBuffer;
      unsigned int    tmpNBuffer;

      // fill extraction parametrs
      THrchEParam tmpEParams;
      tmpEParams.boolFunc = boolFrustFunc;
      tmpEParams.pFuncParams = (void*) i_pCmProps;
      tmpEParams.interval = 40;
      // extract data with parametrs
      m_pHierarchy->Extract(tmpEParams);
      // get buffer of prepared triangles
      m_pHierarchy->GetBuffer(&tmpNBuffer,&tmpBuffer);

      // draw buffer
      glVertexPointer(3, GL_FLOAT, 0, m_vertices);
      glEnableClientState(GL_VERTEX_ARRAY);
      glIndexPointer(GL_INT, 0, tmpBuffer);
      glEnableClientState(GL_INDEX_ARRAY);
      glDrawElements( GL_TRIANGLES, tmpNBuffer * 3,
                       GL_UNSIGNED_INT, tmpBuffer);
}
```

Výpis kódu: Vykreslování datového objektu hierarchické trojúhelníkové sítě

3.3 Konverze k dispozici

3.3.1 Konverze jednoduché t. sítě na hierarchickou

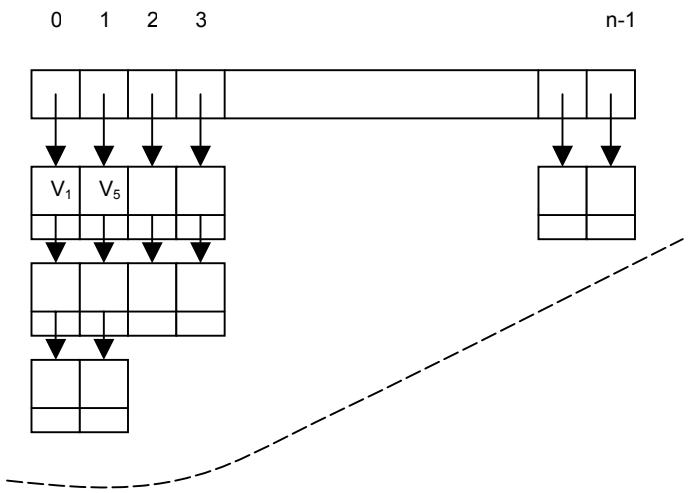
Jediná konverze je zatím mezi datovými objekty jednoduché trojúhelníkové sítě a hierarchické trojúhelníkové sítě CTri2MtrConv. Objektu jsou při konstrukci předány datové objekty CGLTriData a CGLMtrData. Při konverzi se nejprve vyexportují data z objektu CGLTriData a dočasně se uloží ve vnitřních strukturách. Pak se provedou veškeré diagnostické a přípravné operace pro zjednodušování a vytváření hierarchie:

- kontrola manifoldu
- výstavba sousednosti trojúhelníků
- kontrola a úprava souhlasné orientace sousedních trojúhelníků
- výpočet normál trojúhelníků
- generování kvadrik u všech bodů
- generování spojení mezi body a trojúhelníky

3.3.1.1 Diagnostické a přípravné operace

3.3.1.1.1 Kontrola manifoldu $O(N_T)$

Nejprve se vytvoří pole o velikosti počtu vrcholů. Každý element pole je ukazatel na seznam vrcholů s většími indexy, se kterými tvoří hranu v trojúhelníkové síti. U vrcholů v seznamu je ještě počítadlo výskytu hrany. Tento seznam se vytváří při průchodu všech trojúhelníků. Zároveň se inkrementují čítače při pokusu o vložení již existující hrany. Pokud některý čítač přesáhne hodnotu dva, znamená to, že některá hrana je sdílena více než dvěma trojúhelníky. Tato analýza nepovažuje za manifold situaci, kdy se v jednom vrcholu stýkají dvě plochy, protože s tímto stavem je možné dále pracovat. Složitost analýzy je $O(3 \cdot \frac{1}{4} \cdot p_E \cdot N_T)$, kde p_E je průměrný počet hran vycházejících z každého bodu. V praxi se pohybuje okolo hodnoty 6.



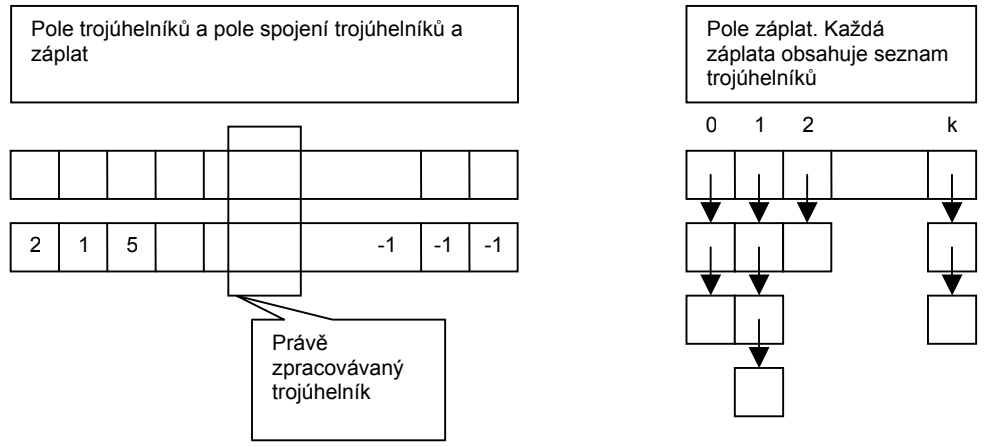
Obr.: Reprezentace hran v poli spojení. Délka seznamů se s narůstajícími indexy zkracuje

3.3.1.1.2 Výstavba sousednosti trojúhelníků $O(N_T)$

Principielně je velmi podobná kontrole manifoldu. Místo čítače je však uveden index trojúhelníka, jenž zapsal do seznamu tuto hranu. Pokud se narazí na již otevřenou hranu, zruší se ze seznamu a obnoví se elementy zúčastněných trojúhelníků v seznamu sousedností. Na závěr se ještě jednou projde pole seznamů a vyberou se zbylé hrany. Sousedi trojúhelníků uložených u těchto hran se nastaví na hodnotu –1. Složitost je podobná jako u kontroly manifoldu, jen se seznam prochází dvakrát.

3.3.1.1.3 Kontrola a úprava souhlasné orientace trojúhelníků $O(N_T)$

Akce spočívá ve vytváření záplat a jejich změně orientace pokud je to nutné. Prochází se všechny trojúhelníky a kontroluje se, zda neincidují s některými záplatami. Mohou nastat čtyři situace. Trojúhelník neinciduje s žádnou, s jednou, dvěma nebo třemi záplatami. Pokud neinciduje s žádnou, sám vytváří novou záplatu. Pokud inciduje s aspoň jednou, kontroluje se souhlasnost trojúhelníka s jedním reprezentantem záplaty. Pak se podle toho, zda nastane špatná konfigurace a podle velikosti zúčastněných záplat, pozmění orientace u nejmenších a všechny se spojí v jednu. Při této analýze se používá pole spojení mezi trojúhelníkem a záplatou inicializovanou na hodnotu –1 u každého trojúhelníku a potom pole záplat se seznamy trojúhelníků, které ji tvoří. Změna orientace záplaty se provádí otočením všech jejích trojúhelníků.



Obr.: Datové struktury použité při úpravě orientace trojúhelníků

3.3.1.1.4 Výpočet normál trojúhelníků $O(N_T)$

Normála trojúhelníku se jednoduše vypočte pomocí vektorového součinu vektorů dvou hran trojúhelníka. Pro potřeby aplikace je nutné je normalizovat. Prochází se pouze všechny trojúhelníky sítě, tedy složitost $O(N_T)$. Normály jsou třeba při výpočtu kvadrik.

3.3.1.1.5 Generování kvadrik u všech bodů $O(N_T)$

Prochází se všechny trojúhelníky a vypočtou se matice A,b a skalár c podle vztahů odvozených v kapitole o zjednodušení. Tyto matice se pak přičtou k maticím každého vrcholu zpracovávaného trojúhelníku. Matice jsou uchovány v poli kvadrik o velikosti počtu vrcholů. Každý element má velikost pouze 10-ti floatových čísel, protože matice A je symetrická.

3.3.1.1.6 Generování spojení mezi body a trojúhelníky $O(N_T)$

Pro snazší orientaci v trojúhelníkové síti je dobré si vytvořit pole spojení mezi vrcholem a libovolným trojúhelníkem, který s ním inciduje. K ostatním pak není problém se dostat pomocí sousednosti trojúhelníků. Spojení se jednoduše vytvoří procházením trojúhelníků a aktualizací indexu trojúhelníku u každého jeho vrcholu. Vrchol tak má vždy index posledního trojúhelníku, který s ním inciduje.

Zjednodušování pomocí kvadrické chybové metriky a výstavba hierarchie probíhá paralelně a každý krok zjednodušení je okamžitě ve formě fragmentu předán do hierarchie.

3.3.1.2 Zjednodušování pomocí kvadrické chybové matice

Akce zjednodušení je zapouzdřena v objektu CSimplification. Do objektu se při konstrukci předávají získané předpočtené informace a základní trojúhelníková síť:

- reference na pole trojúhelníků
- reference na pole vrcholů
- reference na pole sousedů trojúhelníků
- reference na pole spojení vrcholů a trojúhelníků
- reference na pole normál trojúhelníků
- pole kvadrik
- objekt hierarchie, jehož metoda se volá při přealokování pole trojúhelníků

Deklarace třídy se zobrazenými pouze publikovanými metodami.

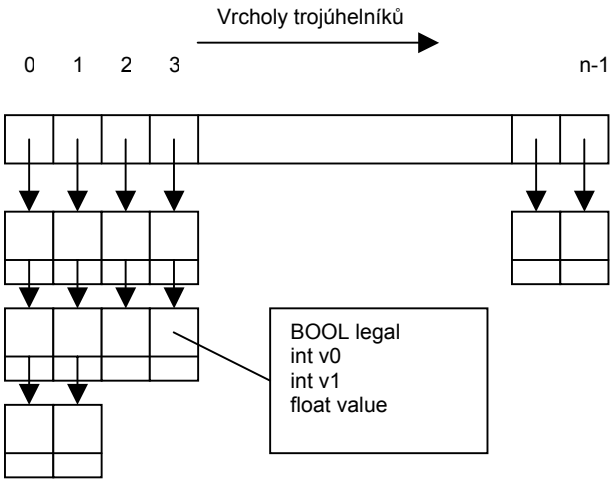
```
class CSimplification
{
public:
     CSimplification(TSimplParam i_params,
                     CFile *i_pLogFile = NULL);
     virtual ~CSimplification();

     void  Init(CProgressCtrl *pProgressBar);
     BOOL  CollapseNext(THrchFINData *o_hrchData,
                        CProgressCtrl *pProgressBar);
     void  GetRmndTri(THrchLFINData *hrchLData);
     void  CleanUp();
}
```

Výpis kódu.: Publikované metody objektu zjednodušování

3.3.1.2.1 Inicializace fronty legálních hran

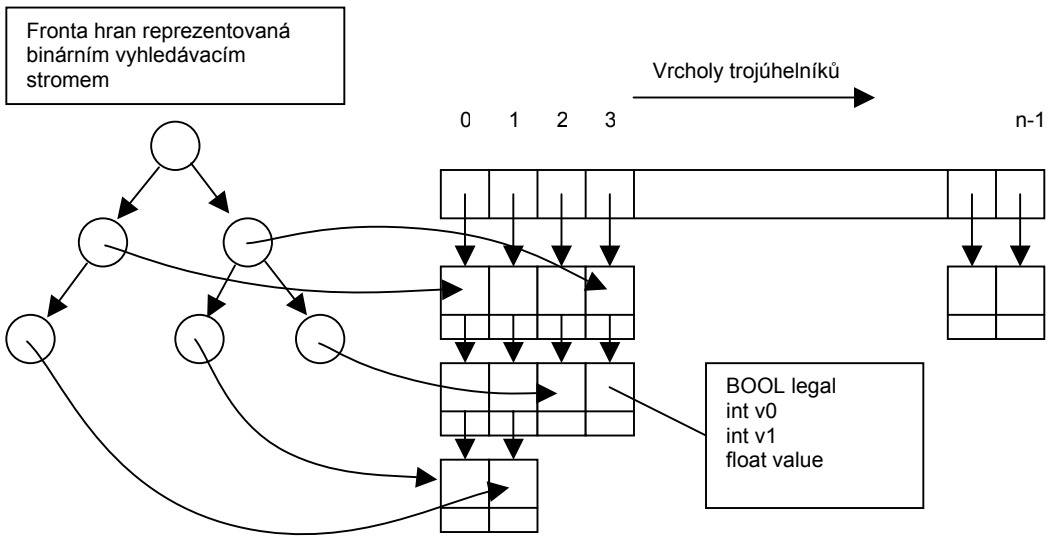
Metoda Init vystavuje frontu hran určených ke kontrakci. Fronta je zapouzdřena v členském objektu třídy CEdgeQueue. Datová struktura popisující frontu hran je vážený binární strom. Při výstavbě se prochází všechny trojúhelníky a jejich hrany se ukládají do pole seznamů hran se stoupajícími indexy vrcholů, již zmiňovaného u kontroly manifoldu. Kromě indexů druhého vrcholu hrany je vedle uložena ještě informace o legalitě hrany, index prvního vrcholu a kvadratická vzdálenost od zúčastněných trojúhelníků. Žádná hrana vycházející z vrcholu na okraji trojúhelníkové sítě není legální. Takto se ochraňují hranice jednotlivých částí trojúhelníkové plochy. Později při zjednodušování ještě k ilegálním hranám přibudou

hrany, které nemohly být z nějakého důvodu kontrahovány. Kvadratická vzdálenost se vytváří součtem kvadrik obou vrcholů a výpočtem ohodnocení v minimálním bodě viz teorie zjednodušování.



Obr.: Reprezentace hran při vytváření fronty hran ke kontrakci

Hrana je současně při vkládání do této struktury hran zařazována také do váženého binárního vyhledávacího stromu, a to podle hodnoty kvadratické vzdálenosti. Do stromu je vkládána pouze reference na element seznamu hran vycházejícího z vrcholu.



Obr.: Fronta hran určených ke kontrakci

Tato struktura umožňuje přistupovat k hranám jak podle vrcholů (ve složitosti $O(\log N)$), tak podle kvadratické vzdálenost od trojúhelníků (ve složitosti $O(1)$). Složitost výstavby struktury je $O(N \log N)$. N_T – procházení všech trojúhelníků a $N_L \log N_L$ - zařazování legálních hran do vyhledávacího stromu. N_L je počet legálních hran.

3.3.1.2.2 Alokace vrcholů a trojúhelníků

Pro zjednodušování a strukturu hierarchie trojúhelníkové sítě je nutné zachovávat všechny trojúhelníky a vrcholy, kterých bylo během procesu dosaženo. Jelikož při každé kontrakci hran dochází k úbytku počtu aktuálních vrcholů o jeden, může celkový počet vrcholů maximálně dosáhnout dvojnásobku počtu vrcholů původní trojúhelníkové sítě.

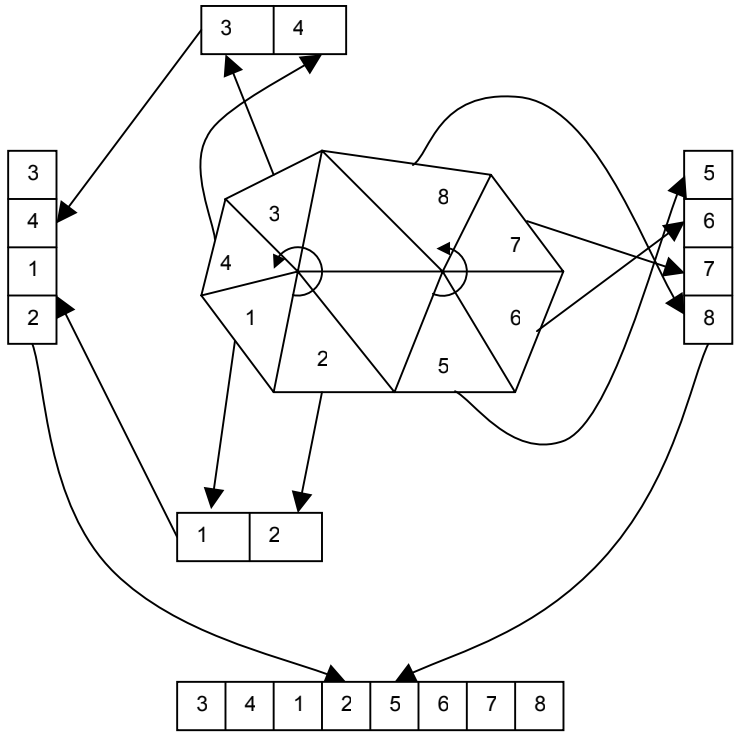
$$|v_{ALL}| \leq 2 \cdot |v_{ORIGIN}|$$

Je tedy možné stabilně alokovat před zjednodušováním dvojnásobně velké pole a po zjednodušení jej oříznout podle potřeby.
S počtem trojúhelníků už je situace poněkud složitější. Při každé kontrakci přibude trojúhelníků o počet zúčastněných na kontrakci snížený o dva vypuštěné trojúhelníky. V praxi se ukazuje, že tento počet leží někde mezi osmi a deseti. Je tedy dobré zvolit si realokační krok – zřejmě ve velikosti počtu původních trojúhelníků. Tak by mělo dojít k realokování pouze několikrát během zjednodušení.

$$|t_{ALL}| \cong 8 \cdot |t_{ORIGIN}|$$

3.3.1.2.3 Průběh zjednodušování

Z fronty legálních hran se vybere hrana s nejnižším ohodnocením. Podle vztahu definovaného v teorii o zjednodušování se určí pozice vrcholu vzniklého kontrakcí hrany. Stěžejní je výpočet inverzní čtvercové matice velikosti 3x3. Zde je naprogramován pomocí adjungované matice. Vychází-li determinant nulový, matice není invertibilní a nový bod se vybere ve středu hrany.
Nyní je důležité naplnit seznam trojúhelníků účastnících se na kontrakci hrany. K tomu slouží již dříve vystavěné pole spojení vrcholů s trojúhelníky. Pro jeden vrchol hrany se zjistí libovolný incidující trojúhelník. Od tohoto trojúhelníku se otáčí dokola kolem vrcholu, až se dojde k jednomu z trojúhelníků společných oběma vrcholům. Přepne se na druhý seznam a začne se od prvního nespolečného točit dokud se nedosáhne původní trojúhelník. Seznamy se spojí a do nového seznamu se začnou plnit trojúhelníky během otáčení se kolem druhého vrcholu. Tento seznam se opět spojí s původním a vznikne tak pás trojúhelníků kolem hrany. To je důležité při kontrole překlopení sousedů.



Obr.: Získání pásu trojúhelníků účastnících se na kontrakci hrany

V seznamu získaných trojúhelníků se dosadí na místo původních vrcholů hrany nový vrchol, vypočtou se normály trojúhelníků a kontroluje se překlopení. To se provede jednoduše kontrolou úhlu svíraného normálami. Pokud úhel přesáhne mezní hodnotu – zde je rovna 90-ti stupňům, kontrakce se zavrhne.

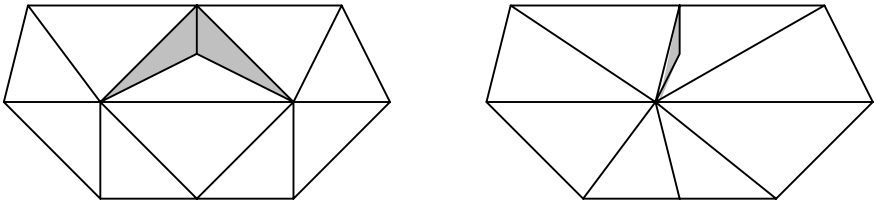
$$\cos(\varphi) = \cos(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \cdot |\vec{v}|}$$

$$\varphi > 90° \Rightarrow \cos(\varphi) < 0$$

$$\vec{u}, \vec{v} \text{ jsou normalizované vektory } \Rightarrow |\vec{u}| \cdot |\vec{v}| = 1$$

$$\text{vektory svírají úhel větší než } 90° \Leftrightarrow \vec{u} \cdot \vec{v} < 0$$

Další kontrola, která s musí provést je kontrola trojúhelníků degenerovaných. Tento případ vznikne při této konfiguraci:



Obr.: Nepřípustná konfigurace bodů

Tato konfigurace je nerozpoznatelná kontrolováním překlopených trojúhelníků, protože trojúhelníky mají správnou orientaci. Je tedy nutné si kontrolovat body hranice této kontrakce a nalezne-li se duplicita jinde než u vrcholů vypouštěných trojúhelníků, kontrakce se zavrhne. Pokud se tedy buď kvůli překlopení nebo špatné konfiguraci kontrakce zavrhne, vyjme se hrana z fronty legálních hran, ponechá se v seznamu hran a označí se jako nelegální.
Pokud je kontrakce přijmuta, vytvoří se nové trojúhelníky a vrchol, opraví se sousednost nových trojúhelníků, vyjmou se z fronty hrany starých trojúhelníků a vloží se do seznamu hrany nových trojúhelníků. Zjednodušený algoritmus je zde:

```
void CSimplification::doCollapse()
{
      vyber první hranu z fronty
      generuj nový bod a jeho kvadriku
      přirav pás trojúhelníků
      nastala špatná konfigurace
            ilegalizuj hranu a vyber ji z fronty
            return
      nastalo překlopení
            ilagalizuj hranu a vyber ji z fronty
            return

      zruš hranu a uprav frontu
      pokud je nutné alokuj více trojúhelníků
      vlož nové trojúhelníky
      oprav sousedy
      připrav data změny
      předej změnu do hierarchie
}
```

Výpis kódu: Zjednodušený zápis operace kontrakce hrany

Proces se opakuje až do doby, kdy je ve frontě hran ještě něco ke zpracování.

3.3.1.3 Zpracování předané změny v hierarchii

Provedená změna se předává do hierarchie formou dvou polí – pole z původními trojúhelníky a pole s novými trojúhelníky. Tato dvě pole víceméně tvoří fragment v orientovaném acyklickém grafu. V objektu CHierarchy je uloženo vnitřní pole o velikosti počtu trojúhelníků. Každý element pole odpovídá referenci na fragment, který obsahuje v podlaze trojúhelník s tímto indexem. Tímto způsobem se jednoduše udržuje informace o příslušnosti každého trojúhelníku. Při inicializaci je toto pole naplněno hodnotami NULL. Vložení změny do hierarchie probíhá podle následujícího algoritmu:

```
void CHierarchy::InsertFragment(THrchFINData i_data)
{
      alokuj místo na fragment
      alokuj místo na horní ukazatele(počet troj. podlahy)
      spočti dolní jedinečné odkazy z pole tfArray
      alokuj místo pro dolní ukazatele
      naplň dolní ukazatele
      pozměň horní ukazatele u všech následníků
      naplň fragment vstupními hodnotami
}
```

Výpis kódu: Vložení změny do hierarchie

Po provedení poslední kontrakce hrany zůstává ještě nezařazena nejjednodušší trojúhelníková síť. Její trojúhelníky tvoří kořenový uzel hierarchie a je nutné s ním nakládat jinak než s ostatními. V CSimplification se tedy získají všechny trojúhelníky, které zbyly metodou GetRmndTris a předají se hierarchii jako poslední data pomocí metody InsertLastFragment. V hierarchii se doplní všichni následníci a reference na kořen se uloží ve vnitřní proměnné.

3.4 Dosažené výsledky

3.4.1 Stabilita

Algoritmus zjednodušování pomocí kvadrické chybové metriky je velice stabilní. Podmínkou je ošetření případů vzniku duplicity trojúhelníků a překlopení trojúhelníků a negativní všechny diagnostické testy.

3.4.2 Paměťová složitost

Výpočet paměťové složitosti je proveden pro tři různé části. Je zde uvedena složitost trojúhelníkové sítě v základní nezpracované verzi, složitost během zjednodušování a vytváření hierarchie a složitost vytvořené hierarchie.

3.4.2.1 Předpoklady

Pro výpočet složitosti se předpokládá „průměrná“ trojúhelníková síť. Taková síť tvoří jeden celek a tvoří uzavřenou plochu. Tedy neuvažují se případy rozdrobených trojúhelníků a

složitost je také vyšší než u neuzavřených sítí, takže skutečná velikost potřebné paměti je nižší.

Předpokládané vztahy

- počet vrcholů N
- počet trojúhelníků v síti 2N
- počet hran zjednodušované troj.sítě 3N
- počet trojúhelníků sídlících jeden vrchol 6
- počet trojúhelníků zjednodušované oblasti 10
- počet trojúhelníků oblasti po zjednodušení 8
- počet kontrakcí N (horní hranice)
- počet trojúhelníků vzniklých kontrakcí 8N (horní hranice)

Velikosti základních datových typů

- uvažuje se jednoduchá přesnost v plovoucí řádové čárce, tedy co číslo, to 4B
- celočíselné hodnoty 4B
- reference 4B

Velikosti odvozených datových typů

- vrchol 12B
- trojúhelník 12B
- soused 12B
- kvadrika (symetrická 3x3, 3x1,1) 40B
- fragment („průměrný“ fragment má 18 indexů troj., 9 referencí, 1 vrchol index, 4 počítadla) 128B
- hrana (dva vrcholy, ohodnocení, 3 reference) 24B

3.4.2.2 Nezjednodušená síť

Uvažuje se základní síť tvořená pouze vrcholy a trojúhelníky.

- N vrcholů
- 2N trojúhelníků

$$m_b = 12N + 24N$$

$$m_b = 36N[B]$$

3.4.2.3 Vytvořená hierarchická síť

Jako u základní se uvažují pouze vrcholy, trojúhelníky. Navíc jsou zde fragmenty a výstupní buffer alokovaný na velikost původní troj. sítě.

- 2N vrcholů
- 10N trojúhelníků
- N fragmentů
- 2N velikost bufferu

$$m_f = 24N + 120N + 128N + 24N$$

$$m_f = 296N[B]$$

3.4.2.4 Vytváření trojúhelníkové sítě

Při vytváření se uvažují dočasné datové struktury a výstupní trojúhelníková síť bez alokovaného výstupního buffer. Složitost se skládá z fixní části a variabilní části. Variabilní se mění během výstavby sítě (jsou alokovány dynamicky oproti fixním, které představují pole alokované pouze jednou na velikost nejhoršího případu), a proto se uvažuje nejhorší případ.

Fixní

- 2N vrcholů
- 10N trojúhelníků
- 10N sousedů
- 10N spojení trojúhelník - fragment
- 2N spojení vrchol – trojúhelník
- 2N referencí na spojení vrchol – vrchol
- 2N kvadrik
- N referencí na fragmenty v hashovací tabulce fragmentů

Variabilní

- 3N hran
- N fragmentů

$$m_{fix} = 12N + 120N + 120N + 40N + 8N + 8N + 80N + 4N$$
$$m_{fix} = 392N[B]$$
$$m_{\text{var}} = 72N + 128N$$
$$m_{fix} = 200N[B]$$
$$m_c = 592N[B]$$

3.4.2.5 Shrnutí

Koeficienty nárůstu paměti lze tedy určit pomocí vztahů:

$$k_f = \frac{m_f}{m_b} = \frac{296}{36} = 8.2$$
$$k_c = \frac{m_c}{m_b} = \frac{592}{36} = 16.4$$

Uveďme si zde nyní jednoduchý příklad trojúhelníkové sítě se 100.000 vrcholy.

$$m_b = 3600000[B] = 3.4[MB]$$
$$m_c = 3.4 \cdot 16.4 = 55.8[MB]$$
$$m_f = 3.4 \cdot 8.2 = 27.9[MB]$$

Paměť potřebná pro vytváření hierarchické trojúhelníkové sítě je tedy poměrně velká. Vzhledem k současné dostupnosti paměti však není problém zpracovávat soubory dat o velikosti milionu bodů.

3.4.3 Dosažené časy

U této aplikace je velice složité rozumně porovnat časy dosažené algoritmem s časy, kterých dosáhl autor metody, protože zde jsou spojovány dvě metody dohromady. Většina časů, které jsou k dispozici, je pouze pro jednu metodu a algoritmy jsou také pro jednu metodu optimalizované. Zde je pokusné srovnání metody simplifikace s tím, že v této aplikaci se časy načítají pomocí čítače a skutečné časy by tedy byly o něco nižší. Test autora proběhl na procesoru R10000 128MGHz se 195MB, což by mohl být srovnatelný počítač s počítačem v testu použitým, tedy Celeron 433MGHz s 256MB paměti. Model použitý k testu byl známý „králíček“ se 69451 trojúhelníky. Test je uveden v [garl0901].

	Inicializace	Zjednodušování	Výsledný
Garland QEM simpl	3,3 s	12,0 s	15,3 s
Tato aplikace	4,9 s	22,3 s	27,2 s

Časy u vytváření hierarchické struktury jsou zanedbatelné vzhledem k časům dosažených při zjednodušování. Pro srovnání zde jsou časy jednotlivých částí :

	Celkem strávený čas
Zjednodušování	27,7 s
Hierarchie	1,0 s

Zajímavější je extrakce. Zde je samozřejmě vidět, že vykreslení kompletně rozbalené hierarchické trojúhelníkové sítě je tvořeno extrakcí a vykreslením nezjednodušované sítě.

	Frame rate	Doba vykreslení	Doba extrakce
Nezjednodušený	1,88	0,53 s	0,0 s
Zjednodušený plně rozbalený	0,58	1,72 s	1,13 s

Pro porovnání lze uvést, že vykreslení kompletně rozbalené hlavy, což může být třetina všech bodů, udržuje frame rate na hodnotě 2,0. Necháme-li králíkovi zobrazit jen kompletně rozbalené ucho, dostáváme se na frame rate 7,0.

4 Závěr

Použité metody se jeví jako celkem zajímavé a dobře v praxi použitelné. Zejména v počítačových hrách, zobrazování velkých objemů dat – CT, MRI rekonstruovaná data, letecké simulace, GIS systémy, atd.

4.1 Metoda obecně

Zhodnocení použité metody začněme nevýhodami, které se objevily během realizace problému. Jedna z velkých nevýhod je nepoužitelnost jakýchkoliv urychlovacích technik grafických knihoven, které jsou implementovány hardwarově na grafických enginech. Jde především o kompresní reprezentaci sítě trojúhelníků do trojúhelníkových pásů a vějířů, čímž je znemožněno uchování dat přímo v paměti grafických karet, což značně urychlí zobrazování.

Výhodou je poměrně snadná implementace obou částí, na dnešní dostupnost operační paměti ne příliš velká paměťová náročnost. Metoda zjednodušování je velice stabilní, dobře aproximuje původní trojúhelníkovou síť. Společná implementace s explicitní multitriangulací je dobře realizovatelná. Metoda explicitní multitriangulace má dobré vlastnosti při rychlé extrakci dat.

4.2 Vlastní implementace

Během implementace postupem času vykrystalizovaly určité milníky, které by se daly při dalším vylepšování eliminovat.

Velké rezervy jsou v realizaci selektivního rozbalování, jež bylo implementováno procházením DAGu do šířky za pomoci inklusivně-exklusivní funkce. Nebyla použita žádná další heuristika, což vede k ne zcela optimálnímu zobrazování. Nejpatrnější je to v okamžiku, kdy se kamera přiblíží příliš blízko povrchu v místě, kde kořenová síť nemá žádný bod. Tam se totiž vůbec nerozbalí. To lze dobře eliminovat použitím heuristiky se vzdáleností od kamery a přímky pohledu. Pak také v souvislosti s rozbalováním vyvstává problém nerovnoměrného rozbalování. Někdy se rozbalí příliš malé trojúhelníky, jejichž diagnostika rodičů rozbalí poměrně velkou plochu na úplně jemné trojúhelníky. Bylo by tedy dobré řadit i fragmenty, které jsou adepty na rozbalování, do fronty, která by priorizvala fragmenty s velkým obsahem (součet obsahů trojúhelníků generovaný během výstavby).

Další podle mého názoru nevýhodná věc je implementace hierarchické struktury plně dynamicky – tedy alokace každého fragmentu samostatně. Tím dochází k defragmentaci paměti. Lepší by bylo použít pole fragmentů a referenční strukturu nahradit indexováním. Problém, který by šlo dále řešit je také velká paměťová náročnost během implementace. Ačkoliv místo, které zabírá již vytvořená hierarchie je 8-krát větší než místo původní sítě, během procesu je alokováno ještě o dost více.

Posledním nápadem, který by zřejmě šlo realizovat by bylo zkombinování této metody z nějakou metodou dělící prostor. Například s clusterizací, aby se zaručila rychlejší lokalizace.

5 Literatura

[garl9901] Garland M.: *Quadric-Based Polygonal Surface Simplification.* Ph.D. dissertation, Computer Science Department, Carnegie Mellon University, CMU-CS- 99-105, 1999.
[garl0101] Garland M., Shaffer E.: *Efficient Adaptive Simplification of Massive Meshes.* Proceedings of IEEE Visualization 2001.
[garl0201] Garland M., Shaffer E.: *A Multiphase Approach to Efficient Surface Simplification.* Proceedings of IEEE Visualization 2002
[hopp9601] Hoppe H.: *Progressive meshes.* ACM SIGGRAPH 1996.
[hopp9801] Hoppe H.: *Efficient Implementation of Progressive Meshes.* Technical report MSR-TR-98-02 Microsoft Research 1998.
[pupp9701] Puppo E., Scopigno R.: *Simplification, LOD and Multiresolution - Principles and Applications.* Eurographics' 1997.
[pupp9801] Puppo E.: *Variable resolution triangulations.* Computational Geometry, Vol.11, N.3-4 1998.
[zara9801] Žára J., Beneš B., Felkel P.: *Moderni pocitacova grafika.* Computer Press, 1998.
[rect6301] Rektorys K.: *Přehled užité matematiky.* Státní nakladatelství technické literatury, 1963.

A Dodatek - programátorská část

A.1 Prohlížeč dat – třída CGLExplorer

Základem je třída CGLExplorer, která funkčně zajišťuje kompletně scénu a od dat vyžaduje pouze “bounding box” a metodu pro vykreslení dat. Umožňuje pohyb kamery, nastavení dat do základní polohy, korekci rotací, různé mody prohlížení, různé projekce a vykreslení celé scény. Explorer se snaží o co nejlepší optimalizaci vykreslování pro OpenGL. Proto jsou veškeré transformace prováděny pouze na matici dat a matice projekce je ponechána v původním stavu. Frustum kamery je tak stabilně nastaveno do intervalu (0,c) ve směru kamery. c je konstanta pro Far (v této aplikaci zvolená jako 10.0). To má za následek, že hloubkový buffer neztrácí zbytečně rozlišovací schopnost způsobenou omezeným rozsahem čísel v plovoucí desetinné čárce.

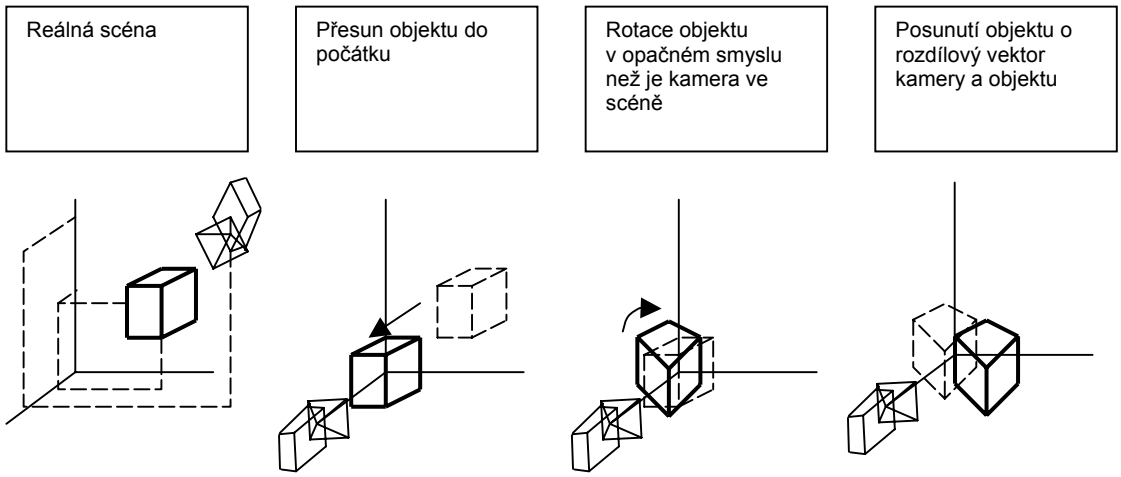
A.1.1 Transformace

Transformace jsou aplikovány na data v tomto pořadí:

- transformace kamery
- transformace frusta
- transformace okna

A.1.1.1 Transformace kamery

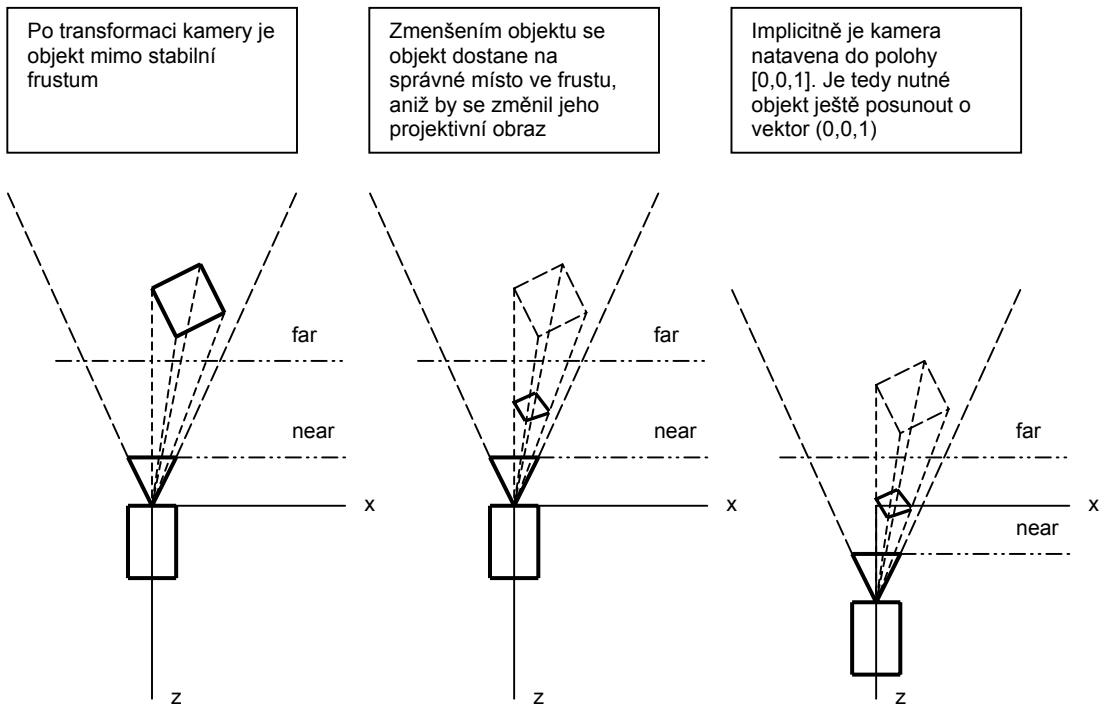
Transformace kamery zahrnují přesunutí objektu do počátku soustavy souřadné, korekci rotace (slouží pro natočení objektu v základní poloze) a inverzní aplikaci polohy a rotace kamery na objekt.



Obr.: Transformace kamery aplikované na objekt v inverzním pořadí (zde je uvažována kamera ještě v počátku s.ř. a jen pro názornější ilustraci je na obrázku posunuta ve směru kladné polosy z)

A.1.1.2 Transformace frusta

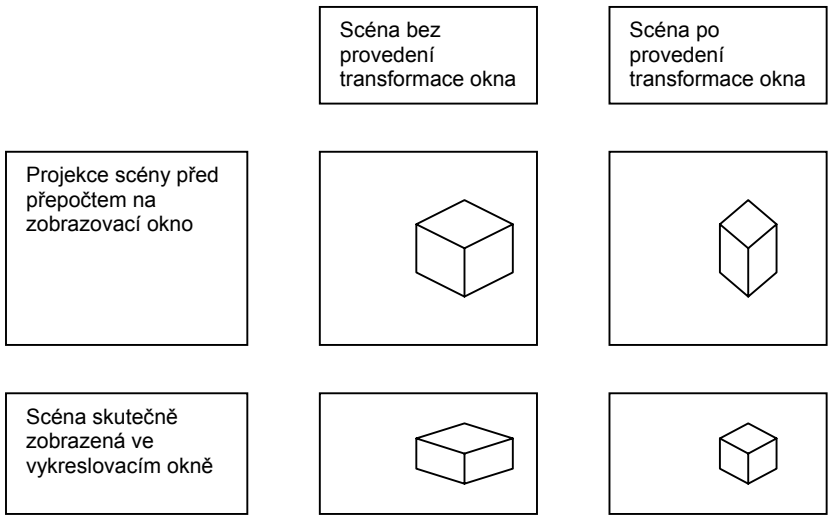
Transformace frusta zmenšuje nebo zvětšuje modifikovanou scénu pro transformaci kamery do stabilního frusta OpenGL kamery a posunuje objekt směrem od kamery o její OpenGL polohu ([0,0,1]).



Obr.: Transformace frusta

A.1.1.3 Transformace okna

Stabilní kamerové frustum je nastaveno při spuštění aplikace na čtvercovou projekční oblast. OpenGL však přepočítává při rasterizaci projekční plochu podle reálného viewportu, který je nastaven v poměru stran vykreslovacího okna. Toto okno není vždy čtvercové a je tedy nutné ještě scénu upravenou transformací kamery a frusta natáhnout(stáhnout) v jednom směru.
K tomu slouží transformace okna. Ta zachovává měřítko v ose menší strany okna a zmenšuje měřítko v ose větší strany.



Obr.: Transformace okna

A.1.2 Nastavení exploreru

Explorer umožňuje nastavení vnitřních vlastností, kterými jsou především:

- mód pohybu pozorovatele (pohyb na kouli nebo volný let)
- mód promítání (perspektivní projekce, paralelní projekce)

A.1.2.1 Mód pohybu pozorovatele

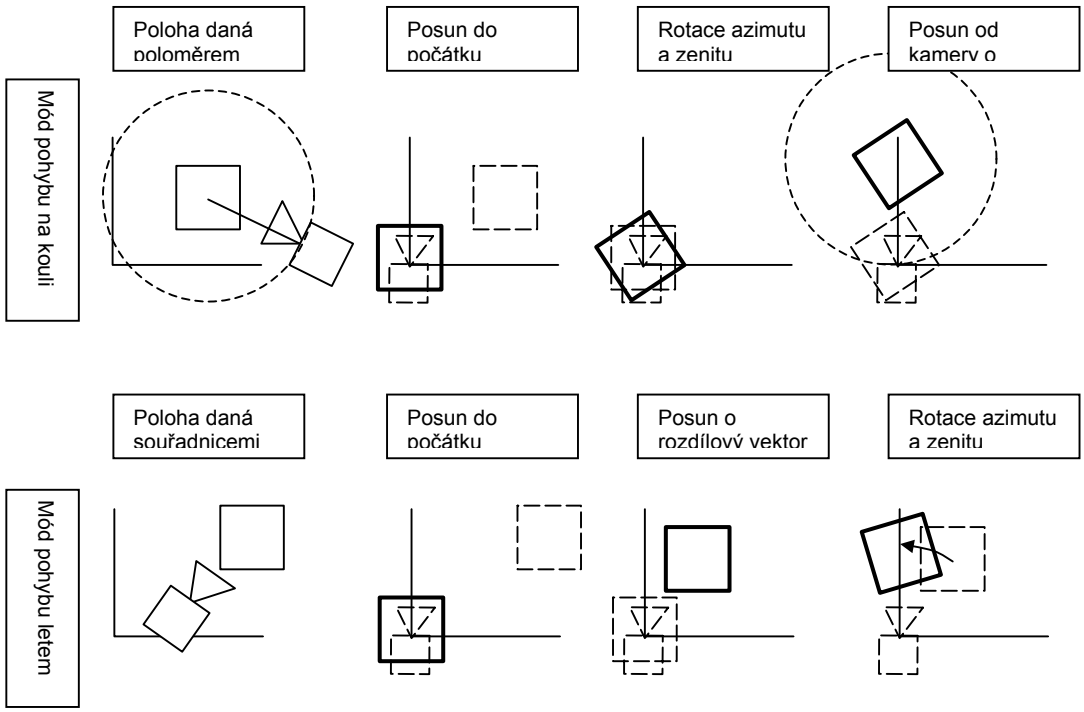
V modu pohybu pozorovatel po kouli jsou na objekt v transformaci kamery aplikovány transformace v pořadí:

- přesun do počátku
- rotace azimutu
- rotace zenitu
- posun o poloměr koule ve směru záporné poloosy z

Střed koule je samozřejmě v centru „bounding boxy“ dat.

V modu pohybu volným letem se uživatel volně pohybuje v prostoru a transformace kamery se provádějí v tomto pořadí:

- přesun do počátku
- přesun o rozdílový vektor kamery a objektu
- rotace azimutu
- rotace zenitu



Obr.: Aplikace transformací kamery pro různé mody pohybu

A.1.2.2 Nastavení výchozí polohy

Kamera se nastavuje do výchozí polohy za pomoci „bounding boxu“ předaného do exploreru objektem dat.

A.1.3 Pohyb kamery

Pohyb kamery je závislý na modu pohybu. U pohybu po kouli je možné pouze ovlivňovat úhel rotace azimutu a zenitu a velikost poloměru. U pohybu letem je více stupňů volnosti. Je možná rotace kamery v azimutálním i zenitálním směru, posun v tangentiálním směru pohledu, posun v binormálovém směru (ve směru kolmém na azimutální rovinu) a posun v normálovém směru (kolmo na směr pohledu a vektor binormály). Senzitivita posunu je opět závislá na bounding boxu. Zde jsou transformační rovnice pro posun v modu pohybu letem:

Směr tangenciální

$$x' = x + d \cdot \sin(azimut) \cdot \cos(zenit)$$

$$y' = y - d \cdot \sin(zenit)$$

$$z' = x + d \cdot \cos(azimut) \cdot \sin(zenit)$$

Směr binormálový

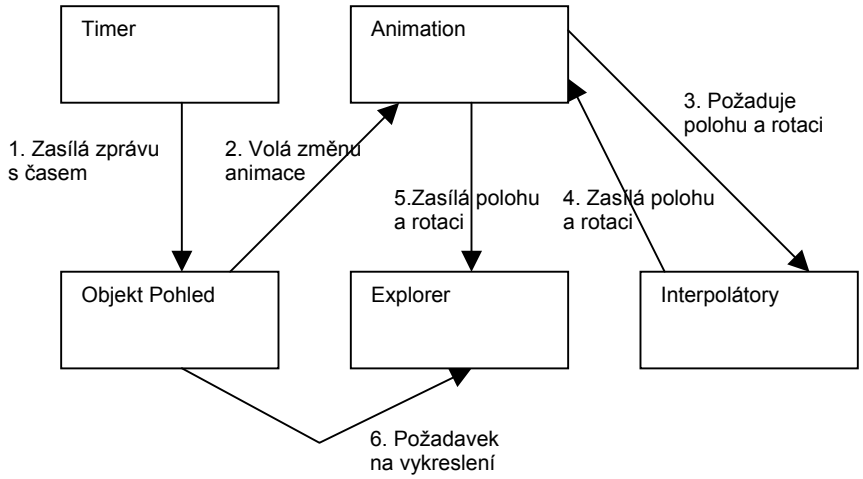
$$x' = x + d \cdot \cos(azimut)$$
$$y' = y$$
$$z' = z - d \cdot \sin(azimut)$$

Směr normálový

$$x' = x + d \cdot \sin(azimut) \cdot \sin(zenit)$$
$$y' = y + d \cdot \cos(zenit)$$
$$z' = z + d \cdot \cos(azimut) \cdot \sin(zenit)$$

A.2 Animace – průlet

Aplikace umožňuje jednoduché animované průlety. Ty zajišťuje třída CAnimation, která v sobě zahrnuje Spline interpolátor polohy kamery a lineární interpolátor natočení kamery ve směru azimutálním a zenitálním.
Animace je založena na nastavování klíčových snímků. Aby bylo ovládání co nejjednodušší, nastaví uživatel kameru do požadované polohy a rotace a stiskem tlačítka informace vloží. Nepožadují se po něm žádná čísla snímků nebo čas. Jen celkový čas animace. Rozdělení vložených pozic se provádí automaticky, a to podle eukleidovské vzdálenosti sousedních poloh kamery. Toto zjednodušení dosahuje uspokojivých výsledků. Rovněž pro zjednodušení byl zvolen lineární interpolátor rotace. Animaci řídí multimediální časovač, který v požadovaném okamžiku přečte přesný časový údaj a zašle jej jako zprávu objektu pohledu. V pohledu se vyvolá objekt animace, který podle času získá z interpolátorů pozice a rotace. Podle získaných údajů se nastaví kamera exploreru a scéna se vykreslí.

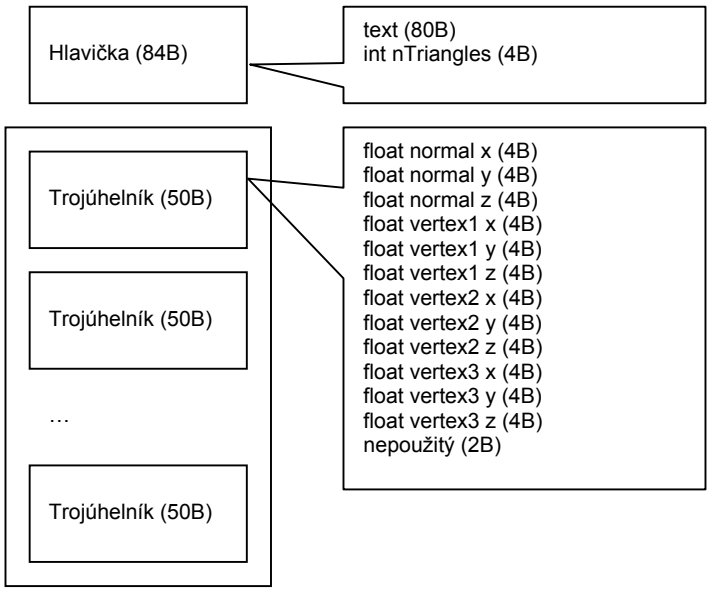


Obr.: Mechanismus animace

A.3 Načítání/ukládání dat k dispozici

A.3.1 Rozhraní stl souborů

Toto rozhraní dokáže v současné době pouze načítat data z binárního stl souboru do jednoduché trojúhelníkové sítě. Binární soubor stl má následující strukturu:



Obr.: Struktura binárního stl souboru

Jelikož data jsou uložena v redundantním formátu (většina vrcholů je společná pro několik trojúhelníků), je nutná určitá optimalizace. Navíc vnitřní formát jednoduché trojúhelníkové sítě využívá trojúhelníky ve formě indexů do pole vrcholů a zde jsou použity jejich skutečné souřadnice.
Data jsou tedy nejprve načtena do bufferu v paměti a je získán jejich „bounding box“. Tento bounding box slouží k inicializaci hashovací funkce. Hashovací funkce zajišťuje rychlou informaci o tom, zda je již vrchol jednou načten nebo ne. V případě, že již jednou byl, vrátí hashovací funkce index nalezeného vrcholu a pokud ne, tak se vytvoří nový vrchol a index je navrácen. Hashovací funkce má následující tvar:

Základ hashovací funkce:

$h = a \cdot v_x + b \cdot v_y + c \cdot v_z$, kde a,b,c jsou určité rozptylovací koeficienty

zvoleno:

$a = \sqrt{2}$

$b = e$

$c = \pi$

teoretické minimum hashovacího základu pak nastává v bodě

$$[x_{\min}, y_{\min}, z_{\min}]$$

a maximum v bodě

$$[x_{\max}, y_{\max}, z_{\max}]$$

cílem hashovací funkce je namapovat interval $(h_{\min}, h_{\max})$ na interval $(0, k * N)$, kde k je koeficient plnění hashovací funkce. Toto namapování se provede následujícím postupem:

$$\begin{aligned} h_{NEW} &= (h(v) - h_{\min}) \cdot \frac{k \cdot N}{h_{\max} - h_{\min}} \\ &= a_{NEW} v_x + b_{NEW} v_y + c_{NEW} v_z - p \end{aligned}$$

Nové koeficienty mají tento tvar:

$$a_{NEW} = a \cdot \frac{k \cdot N}{h_{\max} - h_{\min}}$$

$$b_{NEW} = b \cdot \frac{k \cdot N}{h_{\max} - h_{\min}}$$

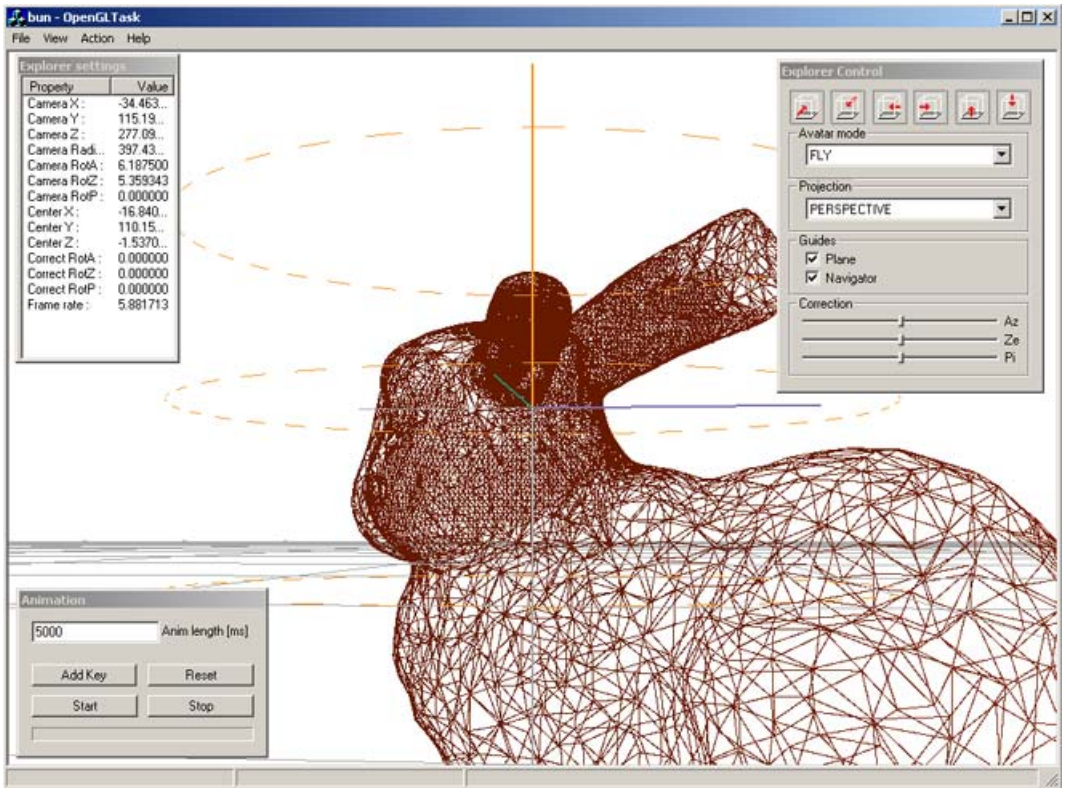
$$c_{NEW} = c \cdot \frac{k \cdot N}{h_{\max} - h_{\min}}$$

$$p = h_{\min} \cdot \frac{k \cdot N}{h_{\max} - h_{\min}}$$

Hashovací funkce tedy generuje klíč k bodů uloženým v seznamu bodů. K nalezení bodu je ještě třeba definovat rovnost dvou bodů, což je velký problém v číselné reprezentaci pomocí plovoucí řádové čárky. Je třeba si zvolit určitou hranici ε, do které je brána hodnota jako totožná. To může vést ke dvěma chybám. Buď se dva logicky totožné body rozdělí ve dva různé nebo, a to je horší případ, se spojí dva různé body do jednoho.
Za pomoci hashovací funkce se tedy vrcholy načítaných trojúhelníků přepočítávají na indexy v poli bodů. Složitost tohoto algoritmu je $O(N_T)$, protože se pouze dvakrát projde pole trojúhelníků.

B Dodatek - uživatelská část

Aplikace OpenGL Task je jednoduchá jednookenní aplikace pro systém Windows. Vyžaduje instalovanou knihovnu OpenGL. Pro první spuštění není třeba žádné instalace. Prohlížeč se skládá z hlavního vizualizačního okna a z několika dialogových oken. V dolní části je stavový řádek informující o průběhu déletrvajících operací. V horní části je standardní roletové menu.



Obr.: Pracovní plocha programu OpenGL Task

B.1 Dialogová okna

Zatím jsou k dispozici tři dialogová okna:

- informační okno exploreru
- příkazové okno exploreru
- příkazové okno animace

B.1.1 Informační okno exploreru

Okno informuje o všech důležitých stavech, jako jsou pozice kamery, poloměr otáčení kamery pro mód pohybu po kouli, střed otáčení kamery – reprezentující střed bounding boxu, korekce tří úhlů a aktuální frame rate (počet snímků za sekundu).

B.1.2 Příkazové okno exploreru

Zde se nastavují módy pohybu kamery a typ projekce kamery, dále je možné přesunout kameru do některé z implicitních poloh, nastavit korekci rotace ve třech úhlech a zapnout nebo vypnout zobrazení navigačních objektů. Tedy horizontální roviny a rotačního válce.

B.1.3 Příkazové okno animace

Umožňuje nastavit aktuální pozici a rotaci kamery jako nový klíčový snímek, zrušit animaci, nastavit celkový čas animace, zapnout / vypnout animaci.

B.2 Stavový řádek

V první části stavového řádku se zobrazují prováděné příkazy nebo informace a v druhém je ukazatel průběhu dlouho trvajících operací.

B.3 Roletové menu

V roletovém menu jsou nabídky pro práci se soubory, pro zobrazení dialogových oken a provedení operací.

B.4 Funkční popis

B.4.1 Načtení souboru

Zatím je možné načítat pouze binární stl soubory. Soubor musí mít příponu „stl“. Průběh načítání se zobrazuje na stavovém řádku. Při načtení souboru se automaticky otevře log soubor se stejným jménem jen příponou „log“. Do tohoto souboru jsou zapsány všechny důležité údaje po celou dobu, kdy je soubor s daty otevřen.

B.4.2 Pohyb po kouli

Jednotlivé pohyby v exploreru se provádí následující kombinací ovládacích prvků:
SHIFT + LMOUSE + myš doprava a doleva
Otočení kamery okolo centra s poloměrem otáčení r ve smyslu azimutu
SHIFT + LMOUSE + myš nahoru a dolu
Otočení kamery okolo centra ve smyslu zenitu.
CTRL + LMOUSE + myš nahoru a dolu
Mění poloměr otáčení kamery

B.4.3 Pohyb letem

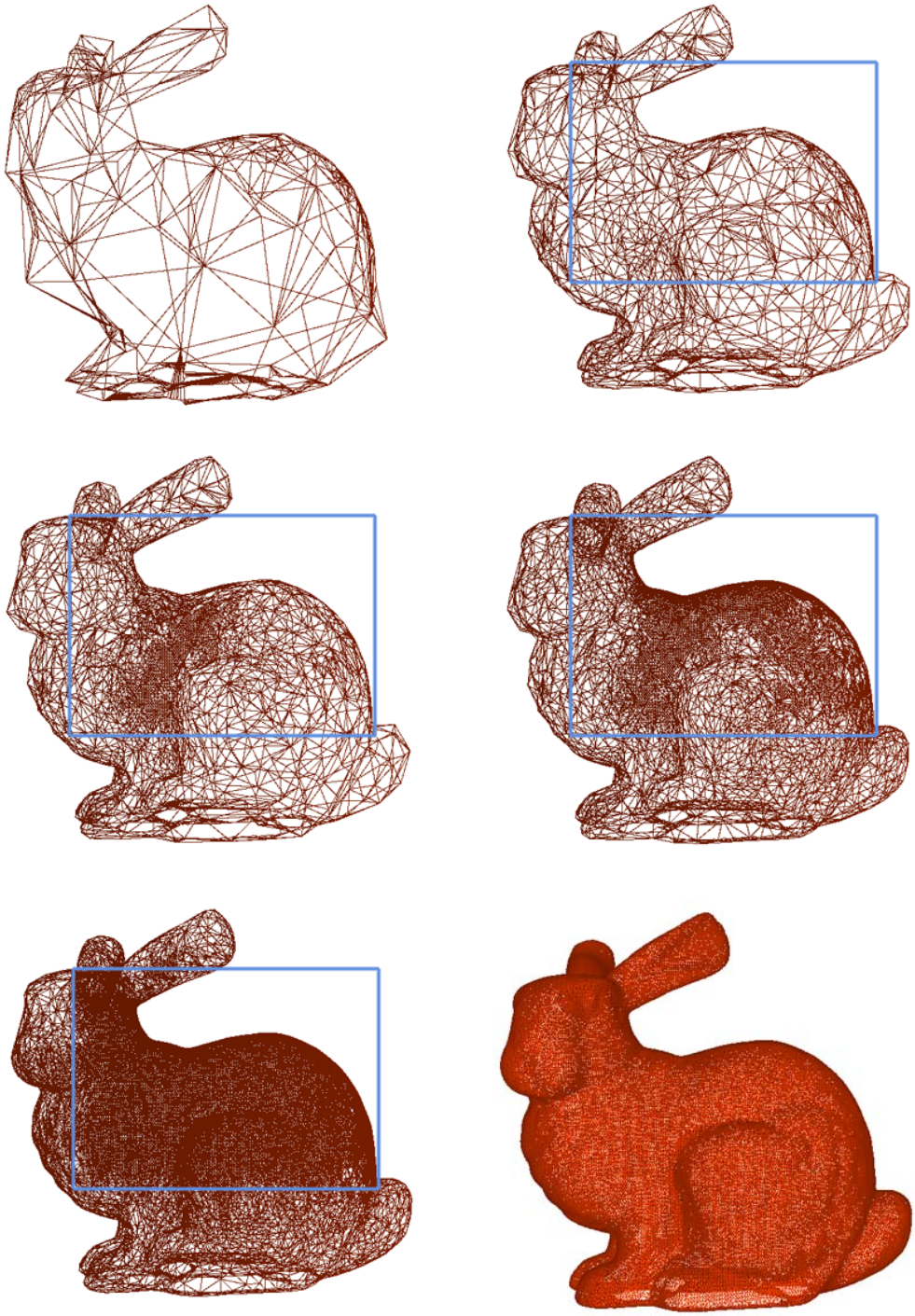
SHIFT + LMOUSE + myš doprava a doleva
Otočení kamery ve smyslu azimutu
SHIFT + LMOUSE + myš nahoru a dolu
Otočení kamery ve smyslu zenitu.
CTRL + LMOUSE + myš nahoru a dolu
Pohyb kamery po tangentě pohledu
SHIFT + RMOUSE + myš doprava a doleva
Pohyb po binormále pohledu

SHIFT + RMOUSE + myš nahoru a dolu
Pohyb po normále ve smyslu pohledu

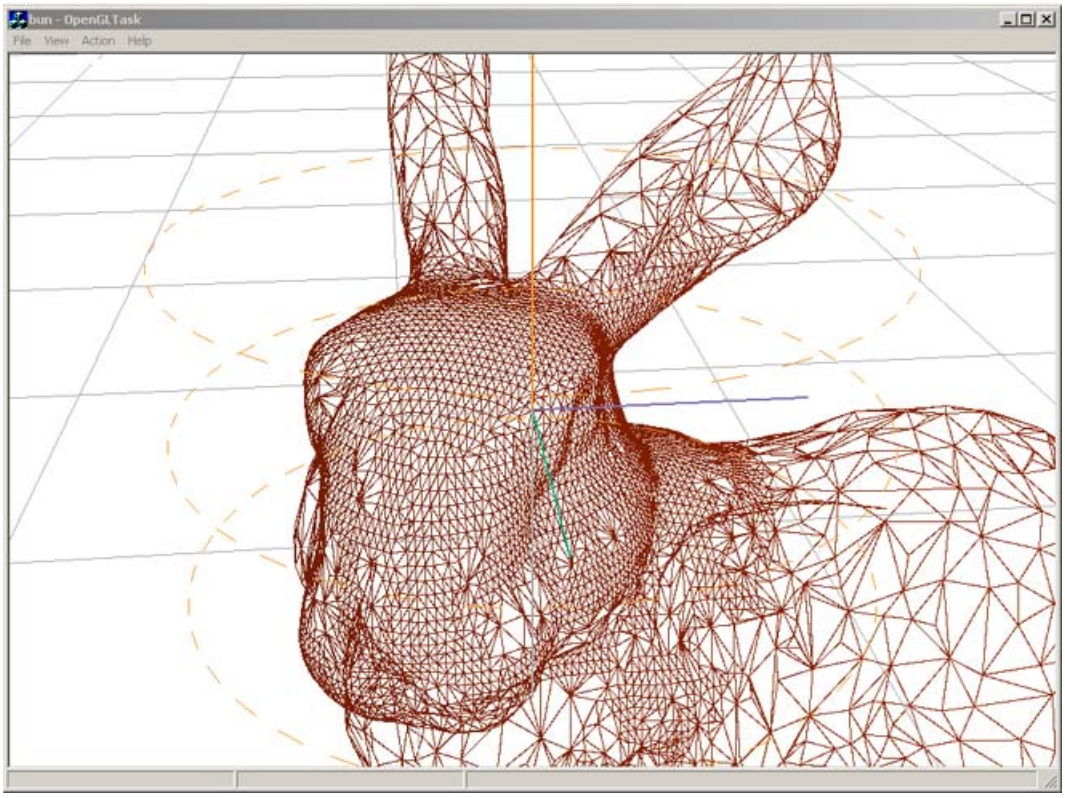
B.4.4 Animace

Explorer provádí animaci tak, že ukládá uživatelem zadané klíčové pozice kamery a pak je interpoluje při požadavku na překreslení z časovače. Časy jednotlivých klíčů jsou odvozeny od eukleidovské vzdálenosti sousedních pozic kamery. Pokud jsou v animaci dva klíčové snímky se stejnou polohou, není možné animaci spustit. Klíčových snímků musí být více než dva.
Upozornění: Při animaci je nutné být v letovém modu. Pohyb po kouli nelze spustit.

C Dodatek - ilustrativní obrázky



Obr.: Model králíka od jeho nejjednodušší sítě se 663 trojúhelníky, přes 2007, 5139,11739,31415 až po kompletních 69415 trojúhelníků. (jsou zobrazeny i neviditelné hrany)



Obr.: Detail hlavy králíka po rozbalení trojúhelníkové sítě při animaci