Západočeská univerzita v Plzni

Fakulta aplikovaných věd

DIPLOMOVÁ PRÁCE

Metody extrakce isoploch pro tetrahedronové sítě a zobrazování neskalárních veličin

Katedra informatiky a výpočetní techniky

Studijní program: IVT – Počítačová grafika

Vedoucí diplomové práce:
Prof. Ing. Václav Skala, CSc.

Autor:
Tomáš Jirka
V Plzni, 2001

Poděkování

Na tomto místě bych rád poděkoval **svým rodičům** za celkovou podporu.

Panu profesorovi **Prof. Ing. Václavovi Skalovi, CSc.** za rady při práci na tomto projektu.

A také **Ing. Martinovi Francovi** za vytištění a svázání této práce v době mého pobytu v zahraničí.

Abstract

This document presents the problems that concern extraction of isosurfaces from the structured as well as unstructured volumetric data and the vector field visualisation. In the theoretical part, the generally used algorithms solving these problems are introduced as well as their pros and cons. The practical section then describes the implementation of chosen algorithms including the advantages and disadvantages of these solutions.

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

V Plzni dne 16.5.2001 Tomáš Jirka

Kapitola 1

Úvod

Extrakce Isoploch a jejich následná vizualizace představuje velmi názornou techniku pro zkoumání skalárních volumetrických dat. Isoplocha totiž vystihuje charakter původních dat, současně z nich však odfiltrovává tu část, která je pro pozorovatele v danou chvíli nezajímavá. Samotná isoplocha však nedává příliš jasnou informaci o průběhu změn v původních datech. K tomuto účelu se používá zobrazení více povrchů, a to buďto najednou nebo postupně.

Současná vizualizace několika isoploch poskytuje oproti postupné možnost přesněji určit velikosti změny v různých částech zkoumaného pole hodnot, protože lze sousední povrchy lépe porovnat a posoudit jejich vzájemnou vzdálenost. Jak se však ukázalo, přináší to s sebou určité nevýhody. Hlavní z nich spočívá v obtížnosti docílit toho, aby si zobrazená scéna zachovala svoji přehlednost, jelikož se jednotlivé plochy mohou navzájem zakrývat. Použití průhledných povrchů a čárových modelů řeší tento problém pouze v případě malého počtu současně zobrazovaných ploch.

Postupné zobrazování jednotlivých isoploch, ať už v podobě animace či v závislosti na interakci uživatele, tyto obtíže odstraňuje. Tento přístup však vyžaduje, aby byly povrchy generovány dostatečně rychle, což je při velkých datových souborech velmi obtížné. Proto je v konkrétních případech nutné zvolit vhodný algoritmus, který bude nejlépe odpovídat požadavkům dané aplikace.

Extrakce isoploch se uplatní i při vizualizaci neskalárních volumetrických dat neboli také vektorových polí. Nepřímou technikou, jak vícerozměrnou informaci zobrazit, je převést ji vhodným způsobem do skalární podoby, ve které pak může být zpracována běžnými metodami pro zobrazování jednorozměrných dat, mezi něž extrakce isoploch patří.

Pro vizualizaci vektorových polí však existuje i mnoho přímých metod. Jejich hlavním cílem je opět poskytnutí přehledné a pro člověka snadno srozumitelné scény, která vystihuje charakter reprezentovaných dat.

Pole vektorů se dnes používají především při zkoumání proudění plynů a kapalin a v různých fyzikálních simulacích. Vektorová data mohou representovat například statické proudění, kdy jsou vektory rychlosti neměnné, či dynamické proudění, při kterém se s časem může měnit nejen směr a délka vektorů, ale i jejich umístění v prostoru [WWW3].

Některé metody vizualizace lze aplikovat bez znalosti toho, co vlastně zpracovávaná data představují. Jiné je potřeba začlenit do určitého kontextu.

1.1 Rozvržení dokumentu

Dokument je rozčleněn do čtyř logických částí. První část (kapitoly 1 a 2) si klade za cíl uvést čtenáře do dané problematiky a seznámit jej s používanými pojmy. Druhá část (kapitola 3), představuje známé algoritmy extrakce isoploch a vizualizace neskalárních dat. V třetí části (kapitoly 4 a 5), jsou popsány implementované algoritmy a další informace, které se jejich realizace týkají. V poslední, čtvrté části (kapitoly 6 a 7) jsou pak zhodnoceny dosažené výsledky a jsou zde také uvedeny návrhy pro další vývoj.

Kapitola 2

Definice pojmů

V této kapitole budou přesněji vymezeny termíny týkající se typu vstupních a výstupních dat a také pojmy, které jsou v literatuře zabývající se problematikou extrakce isoploch používány k popisu vlastností uváděných metod. Některé pojmy však zatím nemají v česky psaných textech ustálený ekvivalent. V těchto případech jsou v závorce uvedeny také původní anglické termíny.

Volumetrická data

Pod tímto pojmem budeme v dalším textu rozumět uspořádanou dvojici (V,W), kde $V = \{v_i \in \mathbf{R}^3, \quad i = 0,...,n-1\}$ je konečná množina vrcholů v oblasti $\mathbf{\Omega} \in \mathbf{R}$ a $W = \{w_i \in \mathbf{R}^k, \quad i = 0,...,n-1\}$ představuje množinu uspořádaných k-tic přidružených k jednotlivým vrcholům. Složky vektoru w_i pro daný vrchol i jsou funkcemi souřadnic tohoto vrcholu $w_{ij} = f_j(v_i), \quad j = 0,..,k-1$. Pro $k = 1$ pak dostáváme soubor *skalárních* volumetrických dat, tak jak je definován například v [Cign96]. Pro $k > 1$ se jedná o *neskalární* data. Tato data, ať už jedno či vícerozměrná, jsou pak dále uspořádána do buněk $\sigma_l, \quad l = 0,..,m-1$, které tvoří rozklad $\mathbf{\Omega}$, a na nichž jsou definovány funkce $\phi_l : \sigma_l \to \mathbf{R}^k, \quad l = 0,...,m-1$, jež interpolují vektory hodnot w_i ve vrcholech jednotlivých buněk. Tyto buňky mají většinou tvar krychle nebo čtyřstěnu a dohromady potom tvoří pravidelnou či nepravidelnou mřížku. Podrobnější dělení geometrických tvarů mřížek navrhli Speray a Kennon ([Sper96] nebo též [Watt92]).

Isoplocha

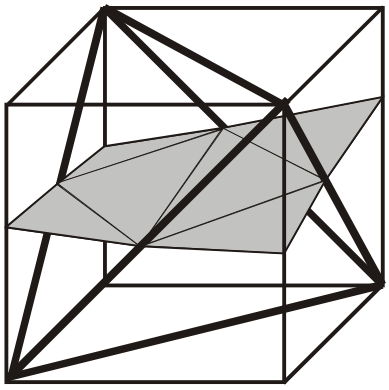
Pokud se jednotlivé funkce ϕ_l na styčných plochách buněk σ_l, na nichž jsou definovány, shodují, získáme spojitou funkci $\phi(p)$, definovanou po částech na množině buněk tak, že:

$$\phi(p) = \phi_l(p), \text{ jestliže } p \in \sigma_l, \forall l = 0,...,m-1$$

Pro danou hodnotu $q \in \mathbf{R}$ a dimenzi $j \in \langle 0, k-1 \rangle$ nazýváme množinu bodů $S(q) = \{p \in \sigma_j : \phi_j(p) = q\}$ isoplochou funkce ϕ pro hodnotu q. Hodnota q bývá označována jako isohodnota či prahová hodnota.

Oblast použití (working domain)

Oblast použití vymezuje typ vstupních dat, na který je možné danou urychlující techniku aplikovat. V mnoha případech totiž nejsou navrhované přístupy dostatečně obecné a lze je tedy použít jen na určitou třídu vstupních dat. Shen a Johnson například v [Shen95] tvrdí, že „teoretická horní hranice počtu buněk protnutých isoplochou je přibližně $O(N^{2/3})$“. Je patrné, že autoři tiše předpokládají určitý konkrétní způsob rozložení tetrahedronů ve vstupních datech. Snadno si lze totiž představit síť tetrahedronů, v níž dokonce většina isoploch protne všech N čtyřstěnů, viz Obr.1.



Obr.1 V tomto případě isoplocha prochází všemi N buňkami.

Vyhledávací přístup (search modality)

Buňky, které je nutné při výpočtu isoplochy navštívit lze nalézt různými způsoby. První ze dvou hlavních přístupů spočívá v procházení geometrického prostoru (*geometric*

approach), druhým způsobem je prohledávání prostoru intervalů, jenž je definován jako množina intervalů mezi minimální a maximální hodnotou ve vrcholech každé buňky. Po nalezení aktivních intervalů, které obsahují hledanou isohodnotu, jsou buňky odpovídající těmto intervalům označeny za aktivní a jsou zařazeny do seznamu, který je nutné při výpočtu isoplochy prohledat (*interval approach)*. Výběr vhodného přístupu zpravidla závisí na struktuře vstupních dat. Obecně lze říci, že se procházení geometrického prostoru většinou využívá u pravidelně uspořádaných vstupních dat, kde je možné využít struktur sloužících k hierarchické dekompozici prostoru, které jsou pro urychlení vyhledávací a klasifikační fáze výpočtu obzvláště vhodné. Intervalový přístup nezávisí na geometrickém rozložení vstupních dat, přináší s sebou však větší nároky na paměť.

Lokální koherence (local coherence)

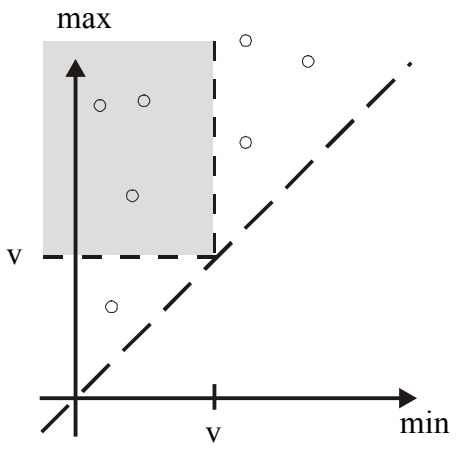
Tento termín označuje návaznost mezi sousedními buňkami. Extrakce jednoho plátu hledané isoplochy z konkrétní krychle či tetrahedronu, poskytuje určitou informaci, která jednak napomáhá stanovit, kterými sousedními buňkami bude isoplocha dále procházet, a jednak snižuje počet nadbytečných geometrických výpočtů, jelikož pláty získané z přiléhajících buněk sdílejí společné vrcholy. Postup, který lokální koherenci využívá, má iterativní charakter a je založen na jistém druhu propagace informací mezi buňkami, což je jedním z důvodů, proč tuto možnost většina algoritmů opomíjí. Řízení procesu šíření informace totiž vyžaduje uchovávání stavu výpočtu a častý přístup do datových struktur s tímto údajem. Nezanedbatelná režie neustálého ukládání a zjišťování stavu výpočtu vede k citelnému zpomalení. Dalším důvodem je skutečnost, že v případě isoplochy sestávající z více oddělených povrchů proces propagace nezaručuje nalezení všech částí.

Globální koherence (global coherence)

Některé algoritmy využívají návaznosti mezi více isoplochami. Část informace získané během extrakce isoplochy odpovídající hodnotě q lze efektivně využít k urychlení výpočtu isoplochy q'. Tato skutečnost je velmi důležitá především v aplikacích, jež zobrazují větší množství povrchů.

Intervalový prostor (span space)

Jedná se o výstižné dvojrozměrné grafické znázornění intervalového přístupu. Na osu *x* je naneseno minimum, na osu *y* maximum (viz Obr.2). Každá datová buňka je pak v grafu representována jedním bodem, přičemž všechny body musí ležet nad osou prvního kvadrantu, jelikož minimální hodnota v buňce je vždy menší, než maximální ($v_{min} < v_{max}$). Ve skutečnosti mohou tyto body ležet také přímo na zmiňované ose ($v_{min} = v_{max}$), tímto případem se však žádný z dostupných algoritmů nezabývá. Úvaha o tom, co by takovýto případ pro výslednou isoplochu znamenal, návrh tří možných řešení a popis jejich výhod a nevýhod lze nalézt v části věnované implementaci, konkrétně v odstavci 4.3.1.



Obr.2 Prohledávání intervalového prostoru – k nalezení isoplochy postačuje navštívit buňky, jejichž obraz leží ve zvýrazněné části grafu.

Kapitola 3

Obecné principy

V první části této kapitoly budou popsány obecné principy platné pro extrakci isoploch z volumetrických dat. Cílem této části je představit známé metody, které se k tomuto účelu používají, jejich výhody a nevýhody. Druhá část se pak obdobným způsobem zaměří na principy a metody zobrazování neskalárních veličin.

3.1 Extrakce Isoploch

V následujících odstavcích budou popsány metody extrakce isoploch z volumetrických dat. Tyto metody se dělí do dvou základních kategorií dle okruhu použití. V první řadě se jedná o algoritmy pro výpočet isoploch výhradně z dat uspořádaných do pravidelné mřížky. Dále jsou to metody, které dokáží pracovat i s daty neuspořádanými. Tomuto dělení odpovídá i rozčlenění této kapitoly.

3.1.1 Pravidelné sítě

Zde budou popsány metody pro extrakci isoploch z volumetrických dat, rovnoměrně uspořádaných do pravidelné mřížky. První dvě, Marching Cubes a Marching Tetrahedra, se vyznačují tím, že během výpočtu hledaného povrchu navštíví všechny buňky vstupního souboru. Třetí metoda používá hierarchické členění geometrického prostoru pro analýzu, která umožňuje odhalit a přeskočit takové oblasti, u nichž je zřejmé, že nebudou danou isoplochou vůbec protnuty.

Marching Cubes

Marching Cubes je pravděpodobně nejznámějším algoritmem pro extrakci isoploch. Oblast jeho použitelnosti zahrnuje pouze volumetrická data uspořádaná do pravidelné pravoúhlé, v ideálním případě krychlové, mřížky, v jejíchž vrcholech jsou uloženy hodnoty nesoucí potřebnou informaci, na jejímž základě se budou požadované isoplochy počítat. V následujícím textu tohoto odstavce bude uvažována krychlová mřížka.

Algoritmus nepracuje s celým objemem dat najednou, ale vždy pouze se dvěma sousedními řezy, kdy jedna ze souřadnic zůstává konstantní. Cyklicky zpracovávanou jednotkou je krychle, jejíž vrcholy tvoří osm sousedních datových vzorků, tedy dvě čtveřice ze dvou přilehlých řezů. Hodnoty v těchto vrcholech jsou nejdříve porovnány s prahovou hodnotou. Je-li hodnota menší než práh, vrchol je považován za vnitřní, v opačném případě za vnější. Takovéto ohodnocení je v podobě osmibitového stavového slova použito jako ukazatel do tabulky, v níž jsou uchovány informace o tom, na kterých hranách mohou v daném případě ležet vrcholy počítané isoplochy. Na nich jsou pak zvolenou technikou nalezeny přesné souřadnice vrcholů plátů hledaného povrchu. Nejčastější metodou pro tento krok je lineární interpolace. Pokud je stěžejním kritériem v dané aplikaci rychlost, je možné místo interpolace umísťovat vrcholy isoplochy do poloviny hran nebo dokonce do vrcholu původní krychle, jehož hodnota je hledanému prahu nejblíže. Takovýto přístup však samozřejmě vede ke zhoršení kvality získaného povrchu. Opakem je použití interpolace vyššího řádu.

Konstrukci zmiňované tabulky, která poskytuje rychlý přístup k informacím o tom, na kterých hranách zpracovávané krychle je potřeba hledat vrcholy extrahované isoplochy, výrazně usnadňuje využití symetrie. Všech 256 položek tabulky je odvozeno od patnácti základních případů, a to jednak rotací a jednak inverzí jednotlivých bitů osmibitového stavového slova, které v podstatě odpovídá vzájemné záměně vnitřních vrcholů za vnější a naopak.

Aby bylo možné plochu hladce vystínovat například Gouraudovou technikou, nelze se obejít bez normálových vektorů ve vrcholech trojúhelníků, jež tvoří vypočtenou isoplochu. Směr normálových vektorů isoplochy je totožný se směrem

vektoru gradientu $\nabla\phi = (g_0, g_1, g_2)$ [1] hodnot původních volumetrických dat v dané oblasti. Vzhledem k této skutečnosti lze normálové vektory ve vrcholech výsledné trojúhelníkové sítě snadno dopočítat lineární interpolací gradientních vektorů ve vrcholech krychle, které však nejsou ve volumetrických datech obsaženy a musí být tedy určeny výpočtem.

Metoda pro nalezení vektorů gradientu ve vrcholech jednotlivých krychlí se nazývá symetrická diference. Složky *x*, *y* a *z* se počítají z hodnot sousedních vzorků, podle následujících vztahů:

$$g_0 = h(i+1, j, k) - h(i-1, j, k),$$

$$g_1 = h(i, j+1, k) - h(i, j-1, k),$$

$$g_2 = h(i, j, k+1) - h(i, j, k-1),$$

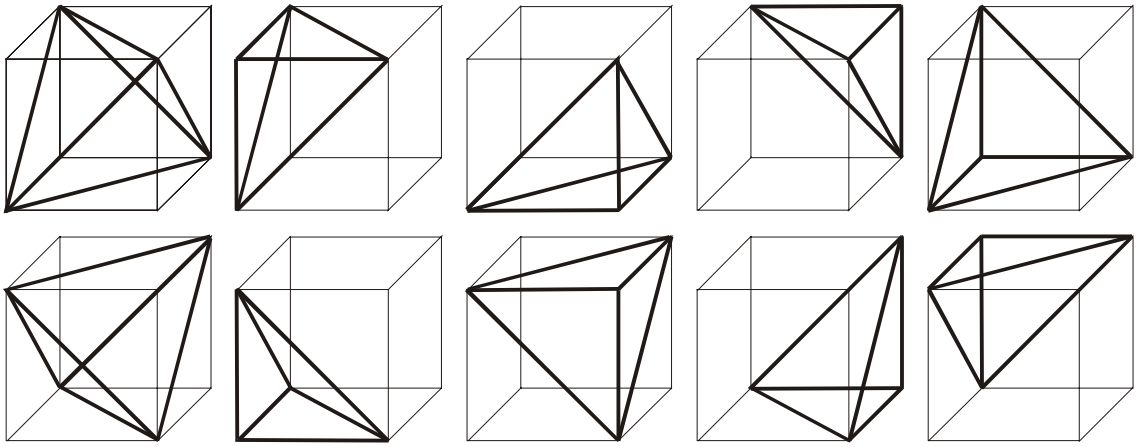
kde i, j a k jsou indexy buněk a $h(i, j, k)$ je hodnota vzorku i, j, k.

Nevýhodou metody Marching Cubes je jednak velký počet generovaných trojúhelníků a také problém dvojznačných stěn a následného vzniku děr, podrobněji popsaný v [Zara98].

Marching Tetrahedra

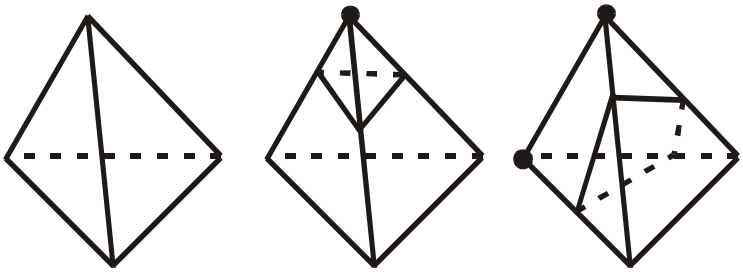
Algoritmus Marching Tetrahedra byl vyvinut ve snaze vyhnout se problémům „dvojznačných stěn“ vznikajících v metodě Marching Cubes a je založen na podobném principu. Krychlová mřížka je však v tomto případě převedena na pravidelnou tetrahedronovou mřížku, a to tak, že je každá krychlička rozdělena na pět symetrických čtyřstěnů. Jak je z Obr.3 dále patrné, lze takovéto dělení provést dvěma způsoby, které je nutné střídat, aby byla zachována návaznost mezi buňkami.

[1] Funkce ϕ je popsána v odstavcích *Volumetrická data* a *Isoplocha* v kapitole 2.



Obr.3 Dvě možná dělení krychle na pět symetrických čtyřstěnů

Algoritmus pak v takto modifikované mřížce sekvenčně projde všechny tetrahedrony, z nichž extrahuje dílčí pláty výsledné isoplochy. Plát může čtyřstěny protínat dvěma různými způsoby. V prvním případě se bude jeden bod nacházet nad isoplochou a tři pod ní nebo naopak a výsledný průnik čtyřstěnu s hledaným povrchem bude mít tvar trojúhelníku, který již není potřeba dále zpracovávat. V případě druhém se budou nad isoplochou nacházet dva body, stejně jako pod ní a výsledný plát bude mít tvar čtyřúhelníku, který je nutné následně rozdělit na dva trojúhelníky. Zřejmou nevýhodou metody Marching Tetrahedra je generování příliš velikého počtu výsledných trojúhelníků.



Obr.4 Možné vztahy mezi tetrahedronem a isoplochou

Obdobu metody Marching Tetrahedra lze použít i v případě nepravidelných tetrahedronových sítí. Procházení všech čtyřstěnů je však velmi neefektivní a ve většině případů výrazně pomalejší než algoritmy, které využívají intervalového prostoru. U nepravidelných sítí navíc vzniká problém s výpočtem normál, jelikož zde nelze využít metodu symetrické diference pro odhad normál ve vrcholech čtyřstěnů.

Octrees

Pro urychlení procesu extrakce isoploch využívají Wilhelms a Van Gelder [Wilh92] stromovou datovou strukturu octree, v jejíchž uzlech uchovávají souhrn údajů o objemu, který reprezentuje podstrom daného uzlu. Obecně mohou tyto údaje zahrnovat jakoukoli užitečnou informaci jako například příznak, zda je celý objem odpovídající danému uzlu prázdný. Popisovaný Wilhelmsův a Van Gelderův algoritmus uchovává v uzlech stromu nejmenší a největší hodnotu, která se v části objemu reprezentované podstromem příslušného uzlu vyskytuje. V první fázi výpočtu, během předzpracování, je vtvořen strom. Hlavní procedura hledání isoplochy začíná ve chvíli, kdy je zadána požadovaná isohodnota. V této části algoritmus prochází vytvořený strom od kořene k listům, přičemž přeskakuje podstromy všech uzlů takových, u kterých extrahovaná isohodnota neleží mezi minimální a maximální hodnotou U_{min} a $U_{\max}$. Ve chvíli, kdy algoritmus dosáhne listu stromu, prozkoumá všech osm buněk, které daný uzel reprezentuje a případně vygeneruje plát isoplochy.

Využití datové struktury Octree výrazně snižuje dobu výpočtu, jelikož umožňuje přeskakování těch částí objemu, které hledaná isoplocha vůbec neprotne. Oblast jejího využitelnosti se však zužuje na data uspořádaná do pravidelných mřížek. Urychlení, které tato metoda přináší, je navíc velmi citlivé na povahu vstupních dat. Různorodé hodnoty ve vstupních datech mohou způsobit, že budou nakonec navštíveny téměř všechny buňky a algoritmus se tak výrazně zpomalí.

3.1.2 Nepravidelné sítě

Je zřejmé, že abychom v daném souboru volumetrických dat nalezli izoplochu odpovídající požadované hodnotě, není zapotřebí procházet vždy všechny buňky datového souboru. Dále popsané metody, které se zaměřují na získávání isoploch z neuspořádaných tetrahedronových sítí, se snaží výpočet urychlit právě co nejvýraznějším snížením počtu zkoumaných buněk. Samotný proces extrakce jednoho plátu isoplochy z odpovídajícího čtyřstěnu se pak již shoduje s postupem uvedeným u metody Marching Tetrahedra v části 3.1.1. Za určitý obecný nedostatek dostupných metod lze považovat fakt, že se žádná z nich nesnaží vyřešit problém odhadu normál ve vrcholech počítané isoplochy z původních dat, jako tomu bylo v případě pravidelné mřížky v metodě Marching Cubes.

MinMax Lists

Tato metoda, kterou vyvinuli a poprvé publikovali Giles a Haimes [Gile90] je postavena na skutečnosti, že konkrétní isoplocha může protínat pouze ty čtyřstěny, pro které platí, že minimální hodnota ze všech čtyř vrcholů daného tetrahedronu je nižší než zadaná isohodnota. Maximální hodnota v daném čtyřstěnu musí naopak hledanou hodnotu převyšovat. V rámci předzpracování tedy Giles a Haimes vytvoří dva seznamy, z nichž první obsahuje minimální a druhý maximální hodnoty všech tetrahedronů. Současně si také zaznamenají, jaký největší rozdíl Δz mezi minimální a maximální hodnotou v rámci jednoho tetrahedronu se v datech vyskytl. Poté hodnoty v obou seznamech seřadí od nejmenší do největší.

Ve chvíli, kdy je poprvé specifikována isohodnota S nebo se změnila oproti své předchozí hodnotě o více než Δz, jsou všechny čtyřstěny, jejichž minimální hodnota se pohybuje v intervalu $(S - \Delta z, S)$, umístěny do takzvaného akčního seznamu, v němž se uchovávají indexy buněk, které by mohla hledaná isoplocha protínat a které je nutné v dalších krocích prozkoumat. Z tohoto seznamu jsou pak odebrány všechny buňky, kterými daná isoplocha neprochází. Zvýší-li se v dalším kroku isohodnota o méně než Δz, pak je akční seznam rozšířen o čtyřstěny, jejichž minimální hodnota spadá do rozmezí mezi starým S a novým S. K tomuto účelu poslouží seřazený seznam s minimálními hodnotami všech čtyřstěnů. Pokud se isohodnota naopak sníží o méně než Δz, pak se do akčního seznamu zařadí ty tetrahedrony, jejichž maximum leží v intervalu mezi novým S a starým S. Pro tento krok je naopak použit seznam seřazených maximálních hodnot všech čtyřstěnů v datovém souboru. Z takto aktualizovaného akčního seznamu se pak opět odeberou všechny tetrahedrony, které extraovaná isoplocha neprotíná.

Metoda MinMax Lists může výrazně urychlit proces výpočtu isoplochy. Má však také určité nevýhody. Hlavním problémem je, že by neměl být rozdíl Δz příliš velký, jinak bude interval $(S - \Delta z, S)$ příliš široký a bude obsahovat velký počet čtyřstěnů. Naopak v případě, že Δz je malé, se počet prohledávaných položek výrazně sníží a metoda dává velmi dobré výsledky. Rozsah Δz však bohužel nelze snadno odhadnout, jelikož k výraznému zvýšení této hodnoty stačí, aby se v celém datovém souboru vyskytovala jediná buňka s velkým rozdílem mezi minimální a maximální hodnotou. Druhou nevýhodou uvedeného algoritmu je časté vkládání a mazání položek

v akčním seznamu, které vyžaduje určitou režii, čímž dále snižuje výkonnost této metody.

Sweeping Simplicies

Shen a Johnson v [Shen95] navrhli metodu, která využívá globální koherence mezi isoplochami a také hierarchické dekompozice dat.

Algoritmus používá dva seznamy. Prvky prvního z nich, tak zvaného *sweep listu*, obsahují ukazatele na odpovídající buňky, jejich maximální hodnoty, podle kterých je seznam seřazen, a příznak. Prvky druhého seznamu nazývaného *min list*, obsahují minimální hodnoty odpovídajících tetrahedronů, podle nichž je seznam seřazen, a ukazatel na pozici této buňky ve sweep listu.

Ve chvíli, kdy je známa požadovaná isohodnota, algoritmus nastaví ve sweep listu příznaky všech buněk, jejichž minimální hodnota je nižší, než zadaný práh. Pokud se nejedná o první hledanou hodnotu, pak algoritmus prochází jen ty buňky v min listu, jejichž minimální hodnota leží mezi starou a novou isohodnotou. V závislosti na tom, zda je nově zadaná isohodnota větší nebo menší než ta předchozí, jsou pak příznaky odpovídajících buněk buďto nastaveny nebo vynulovány. Poté algoritmus prochází sweep listem od první buňky, jejíž maximální hodnota převyšuje aktuální prahovou hodnotu a v buňkách, jejichž příznak je v danou chvíli nastaven hledá jednotlivé pláty výsledné isoplochy.

Jak již bylo uvedeno, algoritmus Sweeping Simplicies využívá hierarchickou dekompozici dat, která je založena na intervalovém přístupu. Data jsou nejdříve rozdělena do N podskupin. Jak bude z dalšího popisu patrné, číslo N je vhodné volit jako mocninu dvou. Toto dělení je považováno za nultou úroveň. Na každé další úrovni jsou podskupiny tvořeny párováním sousedních podskupin z předchozí úrovně, takže na vrcholu získáme jednu skupinu obsahující celou množinu dat. Buňky jsou poté roztříděny do příslušných podskupin na nulté úrovni. Pro každou z těchto podskupin je pak vytvořen její vlastní sweep list a min list, které jsou pak použity během extrakce isoplochy. Přičemž extrakce probíhá vždy na té podskupině, která obsahuje zadanou isohodnotu.

Tento algoritmus sice nezávisí na globálním Δz, rozdělení do podskupin není optimální. Při hledání isoplochy musí být postupně navštíveny všechny buňky obsažené

ve všech podskupinách, do nichž daná isohodnota zasahuje. Tyto skupiny však obsahují intervaly, kterými daná isoplocha neprochází.

3.2 Zobrazování neskalárních volumetrických dat

V této podkapitole budou představeny metody vizualizace vektorových polí. Hlavním cílem těchto metod je především dosažení velké názornosti výstupních scén. Pole neskalárních dat totiž obsahují velké množství informace, která je pak bez předchozího zpracování pro člověka velmi těžce srozumitelná.

Kontrakce

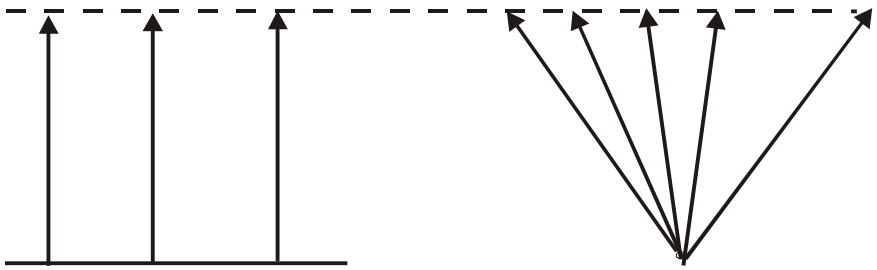
Metoda kontrakce neboli také redukce na skalární hodnoty nezobrazuje vektorová data přímo. Namísto toho zobrazuje pouze určitou skalární vlastnost původních dat. Tou může být například velikost vektoru v daném bodě, skalární součin se zadaným vektorem, zobrazení jen jedné složky vektoru nebo jiná funkce, kterou lze na vstupní data aplikovat. Poté lze již využít například výše popsaných technik pro extrakci isoploch.

Znázornění šipkami

Tato metoda jednoduše znázorňuje vektory tak, jak jsou ve vektorovém poli doopravdy rozloženy. Vektor je v tomto případě znázorněn šipkou, která má počátek na pozici, která odpovídá poloze vektoru ve vstupních datech. Směr šipky je shodný se směrem vektoru. Délka vektoru může být buďto totožná délce původního vektoru nebo je jejím násobkem. To je výhodné v momentě, kdy je vektorové pole velmi husté a normály jsou příliš dlouhé, takže by se při zachování původní délky vektorů výrazně snížila přehlednost zobrazené scény. Ke zvýšení názornosti přispívá tento způsob i v opačném případě, kdy příliš krátké vektory nejsou dostatečně ilustrativní.

Vektory mohou obecně vycházet z jednoho bodu, z bodů na přímce, z bodů rovinné mřížky nebo z bodů na isoploše. Tento typ zobrazení bývá označován jako ježci nebo nitky na pletivu. Pokud jsou vektory zobrazovány tak, že je patrné, z kterého bodu vycházejí, nebo pokud není důležitá jejich orientace ale pouze jejich směr, lze pro jednoduchost zobrazit místo šipek úsečky.

Zjevnou nevýhodou tohoto přístupu je, že se různě dlouhé vektory mohou na obrazovku promítnout tak, jakoby byly stejně dlouhé a naopak (Obr.5).



Obr.5 Pohled z boku (a.) a ze předu (b.) naznačuje, že nemá-li uživatel možnost manipulace se zkoumanou scénou, může být metoda přímého zobrazování vektorů zavádějící.

Sledování částic (particle tracing)

Termín sledování částic zahrnuje několik metod určených pro zviditelnění rozložení pole vektorů na základě interakce s částicemi. Grafické výstupy těchto metod připomínají reálné testy ve větrných tunelech používaní v aerodynamice či pokusy se vstřikováním kouře nebo barviva do proudících kapalin v hydrodynamice [WWW2]. Body vektorového pole jsou postupně integrovány tak, jakoby se jednalo o vektory rychlosti, a výsledné integrální křivky pak charakterizují pohyb těmito vektory reprezentovaný. Z daného místa $x(t)$ a při dané rychlosti $v(x(t))$ se tedy za čas Δt dostaneme do místa

$$x(t+\Delta t) = x(t) + \int_{t}^{t+\Delta t} v(x(t))dt\,.$$

Vypočtené body, které tvoří novou plochu, pro níž je možné integraci znovu opakovat, pak lze s původními propojit úsečkami. Tento cyklus se ukončí, sahá-li poslední vypočtená plocha mimo datovou oblast. Výpočet lze také omezit zadáním maximálního počtu iterací.

S odkazem na [Hans93] uvádí [Zara98] následující rozdělení metod sledování částic podle jejich charakteristik:

Stremlines jsou integrální křivky statického vektorového pole procházející zadaným bodem v daném časovém okamžiku. Jedná se o křivky, které jsou ve všech svých bodech tečné k vektorovému poli. Vzhledem k tomu, že se získávají v jednom časovém kroku, nemají fyzikální paralelu. Počítají se v jednom (tj. statickém) vektorovém poli. Viz také [WWW1].

Streakline je křivka, která vznikne jako dráho částic kontinuálně vstřikovaných z jednoho bodu do vektorového pole. Jinými slovy je to geometrické místo, kde se nacházejí miniaturní částice kapalinu, které prošly zadaným bodem. Fyzikálně odpovídají bublinám, vstřikovaným do proudu kapaliny. Vypočítávají se z mnoha vektorových polí vyvíjejících se v čase.

Path line odpovídá dráze jediné částice v proměnném vektorovém poli. Je to obdoba fotografie svítícího bodu při dlouho otevřené uzávěrce. Vypočítává se z mnoha vektorových polí vyvíjejících se v čase.

Time line vznikne, je-li skupina částic umístěných v jedné přímce v jednom časovém okamžiku naráz uvolněna. Jedná se tedy o skupinu křivek typu *path line*.

Tuft je malý praporek zapíchnutý v nějakém místě. Vznikne buď jako krátká *streamline* v jednom vektorovém poli, nebo jako krátká cesta částice *(particle path)* ve více vektorových polích.

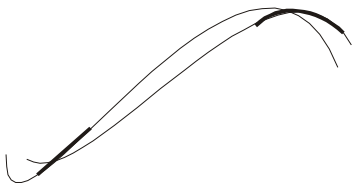
Blob tracing fyzikálně odpovídá sledování vývoje zvoleného konečného objemu kapaliny. Jde o analogii vstřikování barviva do kapaliny. Jde v principu o sledování většího shluku částic.

Při statickém proudění streamline, streakline a path line splývají.

Pásky a Frenetovy trojhrany

Pásky jsou určeny především k detekci zkrutů ve zkoumaném vektorovém poli. Vznikají současným sledováním sousedních křivek typu streamline či streakline (viz

Obr.6). K nalezení a vizualizaci zkrutu slouží také a Frenetovy trojhrany. Jedná se o zobrazení sekvence trojic vektorů, konkrétně tečny $\vec{q}$ k dané křivce, normály $\vec{n}$ a binormály $\vec{b} = \vec{q} \times \vec{n}$, pravidelně rozmístěných na některé z uvedených křivek křivek.



Obr.6 Zkrut ve vektorových datech

Vektorové glyfy

Pod tímto pojmem se skrývá grafický objekt, který něco representuje. Některé glypfy mohou znázorňovat i více vlastností naráz. Takové glyfy se pak mohou skládat z více částí, které se lišší tvarem, barvou a velikostí v závislosti na datech, jaká data se za glyfem skrývají. Uživatel může také glyf interaktivně posunovat a přenášet do míst, která jsou pro něho zajímavá a pozorovat změnu glyfu, podle které poté zjistí vlastnosti dat v dané oblasti.

Integrální křivková konvoluce

Line Integral Convolution (LIC) je moderní metoda vizualizace jak dvojrozměrných, tak i trojrozměrných vektorových polí. Tato metoda je založena na konvoluci náhodné vstupní textury nebo šumového obrázku podél křivek stremline stacionárního proudění. Na vypočtený obrázek, který názorně ukazuje směr proudění, lze dále upravovat aplikací dalších metod. Pomocí umělého obarvování lze například zobrazit nějakou doplňující skalární informaci, případně lze dále upravovat oblasti, které nejsou příliš názorné například proto, že v daném výseku vektorového pole bylo proudění příliš pomalé. Tato metoda je detailněji popsána na [WWW4].

Kapitola 4

Realizace – Extrakce Isoploch

V této kapitole bude popsána implementace vybraných metod extrakce isoploch z nepravidelných tetrahedronových mřížek, jejichž vrcholy mohou být ohodnoceny jak skalární hodnotou, tak i vektorem. Dále pak použité datové struktury, problémy, které s extrakcí isoploch souvisejí, případně implementační záležitosti, kterým bylo nutné věnovat zvláštní pozornost, a také to, do jaké míry a jakým způsobem jsou tyto problémy v současnosti vyřešeny.

4.1 Seznam implementovaných metod

Na tomto místě budou pouze vyjmenovány algoritmy pro extrakci isoploch, které byly pro implementaci zvoleny, a jejich velmi stručná charakteristika. Podrobný rozbor realizovaných metod bude uveden dále v podkapitole 4.4.

Pro implementaci byly zvoleny následující tři metody:

i. **Naivní metoda** Tento algoritmus lze jinak také označit za metodu brutální síly. Jelikož jsou při každém použití tohoto algoritmu navštíveny všechny buňky původní sítě, lze jej považovat za obdobu výše popsané metody Marching Tetrahedra (viz podkapitola 3.1.1), která se jinak používá pouze pro pravidelné mřížky.

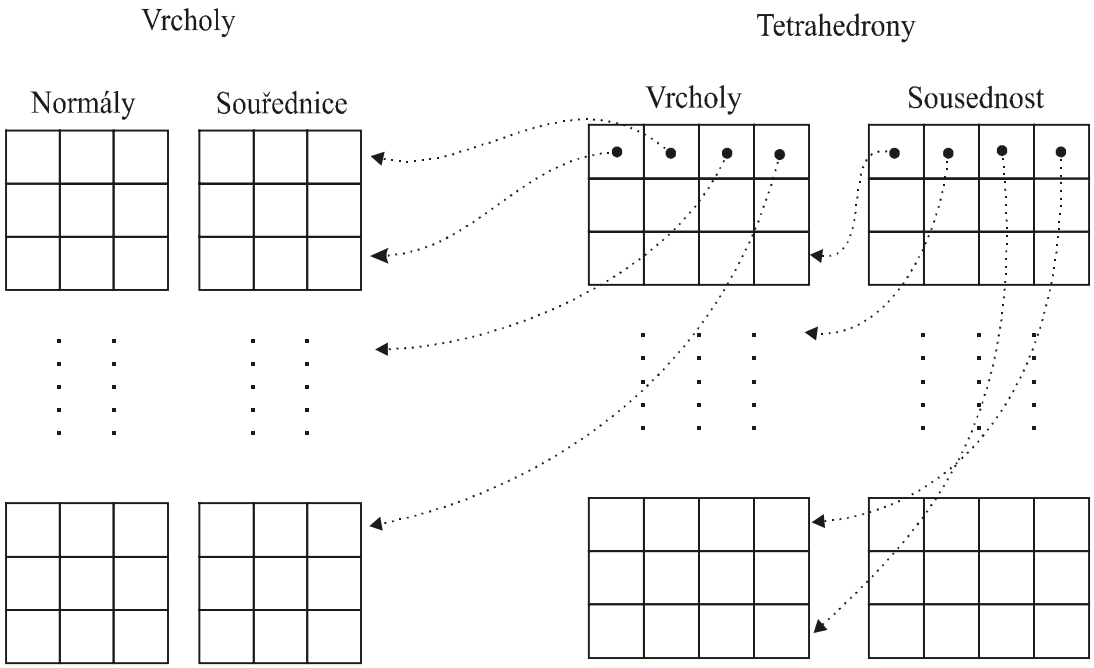
ii. **Prohazování polí** Jedná se o modifikaci naivní metody. K urychlení dochází pouze při extrakci většího počtu isoploch najednou. Její pracovní název zní *Swapping Arrays*.

iii. **MinMax Lists** Modifikace algoritmu popsaného v podkapitole 3.1.2. Tato metoda byla implementována tak, aby umožňovala paralelní výpočet.

4.2 Datové struktury

V následujících odstavcích budou prezentovány použité datové struktury, a to jak vstupní tak i výstupní.

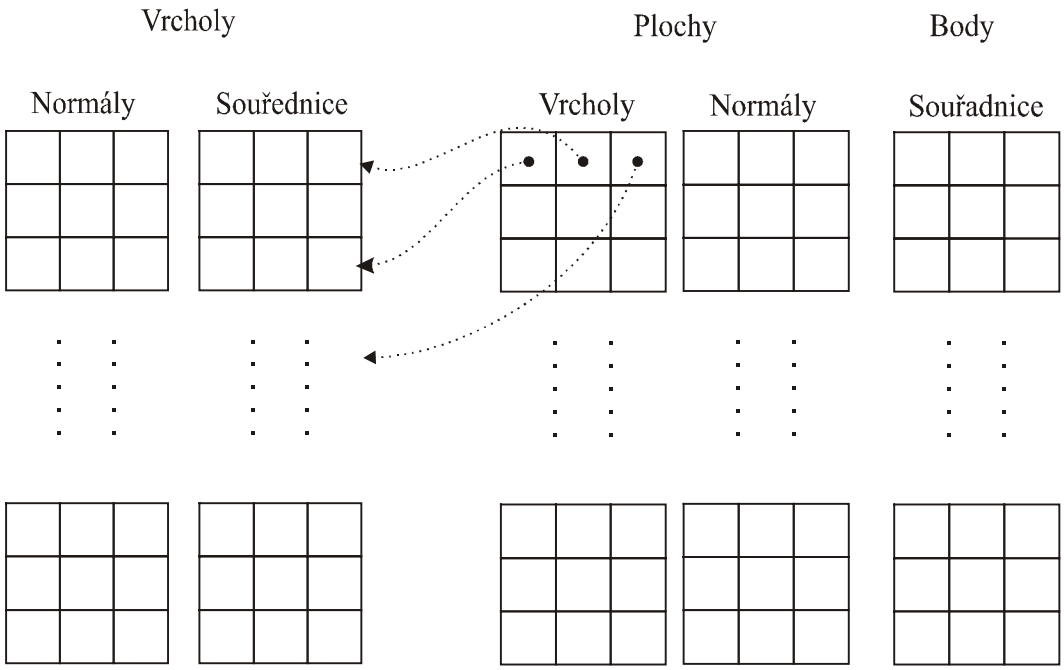
4.2.1 Vstupní datová struktura – tetrahedronová síť



Obr.7 Využívaná část datové struktury MVE Tetrahedra

Datová struktura pro vstup tetrahedronové sítě se v MVE nazývá Tetrahedra 2. Na obrázku je znázorněna její část používaná v tomto programu.

4.2.2 Výstupní datová struktura – soubor isoploch



Obr.8 Datová struktura Surfaces odvozená od struktury Triangles

Výstupní datová struktura se nazývá Surfaces a byla vytvořena modifikací struktury Triangles, přidáním pole bodů, které lze pak v rendereru zobrazit. K odlišení jednotlivých isoploch se také používá pole pro ukládání statutu trojúhelníků. Toto pole není z důvodu přehlednosti na přiloženém obrázku znázorněno.

4.3 Přidružené problémy a jejich řešení

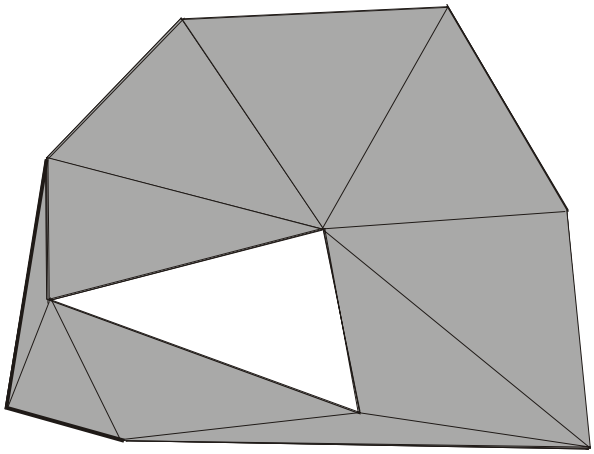
Při realizaci různých metod se zpravidla objevují problémy, které nejsou přímou součástí implementovaných algoritmů, ale které je pro dosažení správné funkce přesto nutné řešit. Následující odstavce se věnují třem konkrétním problémům tohoto typu. Zatímco první dva jsou algoritmického rázu, třetí má charakter spíše implementační.

4.3.1 Isoplocha s bublinami

Žádný z dostupných algoritmů pro extrakci isoploch z nepravidelných tetrahedronových sítí se nesnaží řešit situaci, kdy se budou hodnoty v některých vrcholech určitého čtyřstěnu přesně rovnat hledanému prahu. Jsou-li tyto hodnoty uloženy jako reálná čísla

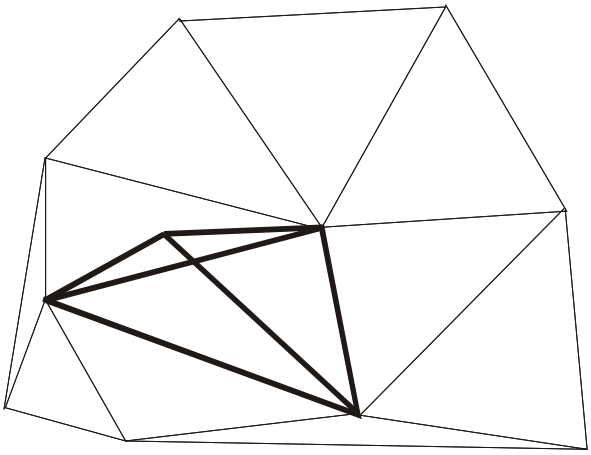
a jestliže navíc byly určeny výpočtem či měřením s velkou přesností, je pravděpodobnost výskytu takového čtyřstěnu velmi malá. Pokud jsou však vrcholy ohodnoceny celými čísly nebo reálnými čísly s nepříliš velkou přesností, může se takový případ snadno objevit.

Jak danou situaci řešit není jednoznačné. Algoritmus, který se tímto problémem explicitně nezabývá, ponechá v takovémto místě isoplochy díru, což lze z určitého hlediska považovat za pravý opak intuitivně očekávaného řešení. Hledáme-li totiž co nejpřesnější aproximaci isoplochy pro danou hodnotu q, není vhodné, aby v místě, kde se hodnoty datových vzorků přesně rovnají hledanému prahu, zela v isoploše díra.



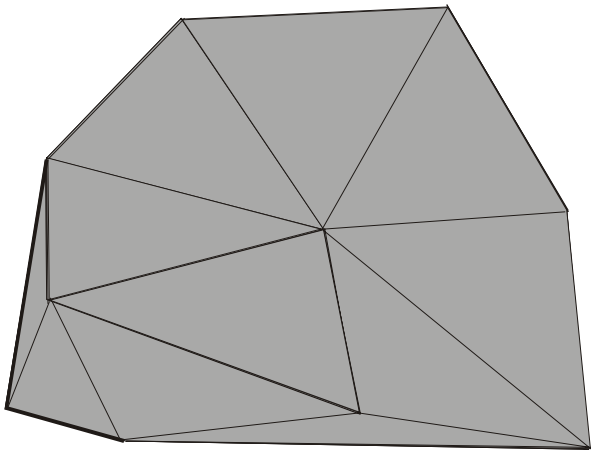
Obr.9 Neúplná isoplocha

Z tohoto pohledu se jako optimální řešení jeví přidání všech čtyř trojúhelníků tvořících daný tetrahedron do výsledné isoplochy. Zmíněný způsob má však opět i své nevýhody. Především je to skutečnost, že na isoploše vznikne „bublina“ s určitým objemem (viz Obr.10), což by mohlo být v některých technických aplikacích na závadu. Současně tím také přicházíme o možnost, uchovávat informaci o sousednosti jednotlivých trojúhelníkových plátů dané isoplochy, jelikož jak trojúhelníky, které tvoří vzniklou bublinu, tak i trojúhelníky s bublinou sousedící, sdílejí každou ze svých hran s dvěma či dokonce více dalšími pláty. Bublina také zvyšuje teoretický horní limit počtu trojúhelníků, které mohou být z jednoho tetrahedronu vygenerovány ze dvou na čtyři, což částečně komplikuje správu přidělování potřebné paměti. Podrobněji bude tato problematika rozvedena v odstavci 4.3.3.



Obr.10 Isoplocha s bublinou, která vznikla přidáním celého tetrahedronu

Jako třetí možné řešení, a na první pohled nejlepší, se jeví využití lokální koherence mezi sousedními buňkami vstupních volumetrických dat, k určení toho, která ze čtyř stěn problematického tetrahedronu by z globálního pohledu v isoploše chyběla a tvořila tak díru. Do výsledného povrchu by pak byla přidána pouze tato stěna. Popisovaný přístup však selhává ve chvíli, kdy se v daném tetraheronu sbíhá více povrchů rozvětvené isoplochy. Navíc v momentě, kdy je inkriminovaná hodnota společná rozlehlejší oblasti vstupních dat, takže se hledanému prahu rovnají i hodnoty ve vrcholech všech okolních tetrahedronů, získává tato metoda rekurzivní charakter, což by v extrémních případech mohlo vést k neúnosným paměťovým i časovým nárokům.



Obr.11 Ideální případ

Aby lépe vynikly vlastnosti jednotlivých metod a abychom mohli uvedená řešení lépe porovnat, podívejme se krátce na to, jak se jednotlivé algoritmy zachovají ve chvíli, kdy se budou danému prahu rovnat hodnoty všech vzorků vstupních dat. Zatímco první

přístup by v takovémto souboru vůbec žádnou plochu nenalezl, druhý zmiňovaný algoritmus by na výstup vygeneroval celé původní těleso rozdělené na čtyřstěny tak, jak byla rozdělena i původní tetrahedronová struktura. Veškeré trojúhelníky tvořící vnitřní plochy, tedy ty, které nejsou součástí pláště původního datového objektu, by se ve výstupním souboru objevovaly dvakrát. Výstup třetího algoritmu lze v takovémto případě těžko odhadnout, neboť by se v datech nevyskytovala žádná oblast, z níž by mohl algoritmus jakkoli určit, které další trojúhelníky nejvíce vyhovují pro přidání k isoploše. V případě rozsáhlého datového souboru by navíc pravděpodobně došlo k zahlcení programu v důsledku hlubokého vnoření rekurze, jak bylo naznačeno v předchozím odstavci.

Po zhodnocení těchto vlastností byl pro implementaci zvolen druhý z popisovaných přístupů. Ověření jeho chování právě v popisovaném extrémním případě lze dosáhnout použitím funkce *Constant Value* v nastavení modulu (viz PŘÍLOHA A).

4.3.2 Výpočet normál

Aby bylo možné názorněji demonstrovat rozložení hodnot ve volumetrických datech a také kvůli možnosti použít lepší stínovací metody, je nezbytné doplnit výslednou isoplochu o normály ve vrcholech jednotlivých trojúhelníků. Tyto normály nelze spočítat přesně. Většinou se za normálu v určitém vrcholu trojúhelníkové sítě považuje aritmetický průměr normál jednotlivých trojúhelníků, které tento vrchol sdílí. Tak je tomu například v algoritmu pro stínování povrchů Gouraudovou metodou. Naším hlavním kritériem však bylo, aby normály ve vrcholech nezávisely pouze na tvaru výsledné plochy, nýbrž aby byly odvozeny z hodnot vstupních dat. Důvodem je skutečnost, že tyto normály nebudou sloužit primárně ke zvýšení kvality stínování isoploch, ale k tomu, aby je bylo možné společně s isoplochou zobrazit v podobě vektorů či úseček, a poskytnout tak pozorovateli lepší představu o průběhu změny ve vstupních volumetrických datech. Ve skutečnosti se tedy nejedná přímo o normály plochy, tak jak je chápeme v geometrii. Takové normály, jak již bylo řečeno, nelze přesně vypočítat, ale spíše pouze odhadnout různými technikami. Přiléhavější než pojem „normála ve vrcholu isoplochy“ se v tomto případě jeví spíše termín „vektor gradientu původních volumetrických dat v daném bodě“. Avšak vzhledem k tomu, že

v ideálním případě by měly být oba tyto vektory rovnoběžné [Zara98], a také proto, že v souvislosti se zobrazováním trojúhelníkových sítí se zpravidla používá spíše termín normála, přidržíme se v dalším textu tohoto pojmu.

Výpočet z neskalárních dat

Zde je situace velmi snadná. V jednotlivých vrcholech původní tetrahedronové sítě totiž máme k dispozici celé vektory hodnot, z nichž se pak kontrakcí (viz kapitola 3.2) získává hledaná isoplocha. Pro nalezení normály v daném vrcholu isoplochy pak stačí interpolovat vektory z příslušných vrcholů tetrahedronové sítě.

Výpočet ze skalárních dat

Samotné normály ve vrcholech isoploch počítáme stejně jako v případě neskalárních dat interpolací vektorů z odpovídajících vrcholů původní sítě čtyřstěnů. V tomto případě však těmito vektory nedisponujeme a musíme je tedy nejprve odhadnout. Z tohoto důvodu je výpočet normály při extrakci isoplochy ze skalárních dat výrazně složitější.

Implementovaný postup vychází z principů symetrické diference (viz podkapitola 3.1.1, podrobněji též [Zara98]), která je však v podobě, jak byla definována, aplikovatelná pouze na rovnoměrné mřížky. Stejně jako v metodě symetrické diference se i zde dopočítává vektor gradientu skalárních dat z hodnot okolních buněk. Při použití nepravidelné mřížky je však nutno tento algoritmus výrazně modifikovat.

V první fázi si program vytvoří seznam všech čtyřstěnů, které sdílí zkoumaný vrchol. V tomto kroku je využívána lokální koherence mezi buňkami. Proto také program umožňuje výpočet normál pouze v případě, že vstupní data obsahují informaci o sousednosti jednotlivých buněk. V dalším kroku pak tímto seznamem prochází a z vrcholů, se kterými bezprostředně sousedí, dopočítává výslednou normálu podle následujícího postupu:

- Vypočte se směrový vektor od zkoumaného vrcholu, k právě zpracovávanému bezprostředně sousednímu vrcholu.
- Spočítá se délka tohoto vektoru a vektor je znormalizován.
- Vyjádří se rozdíl mezi hodnotou, která přísluší právě zpracovávanému vrcholu, a hodnotou zkoumaného vrcholu.
- Tento rozdíl je pak vydělen dříve vypočtenou délkou směrového vektoru.

Za výslednou normálu je pak považován vážený součet takto vzniklých jednotkových vektorů, kde vahou je hodnota vyjádřená ve čtvrtém kroku.

Jak se ukázalo, tento algoritmus sice poskytuje informaci o směru gradientu skalárních dat, ta však není dostatečně přesná, což je patrné při pokusu použít vypočtené vektory jako normály pro stínování isoplochy metodou Gouraud. Největší chyba se objevuje v bezprostřední blízkosti okraje datové množiny. Hlavní problém spočívá v tom, že tato metoda zohledňuje geometrickou strukturu původní tetrahedronové sítě. Tomuto jevu sice nelze při práci s nepravidelnou sítí zcela zabránit, je však zapotřebí jej co nejvíce potlačit. Autor v současnosti pracuje na algoritmu, který by měl vliv rozložení vzorků dat snížit. Tento algoritmus však zatím není dostatečně stabilní, aby mohl být zahrnut do této práce. Druhou nevýhodu implementovaného algoritmu je jeho relativní časová náročnost (viz Kapitola 6).

4.3.3 Správa paměti

Z odstavce 4.3.1 je patrné, že jeden tetrahedron může k výsledné isoploše přispět čtyřmi trojúhelníky. Je-li tedy zapotřebí vypočítat M isoploch z datového souboru čítajícího N tetrahedronů, je možné dopředu určit maximální počet trojúhelníků X, z nichž se může množina vyextrahovaných povrchů skládat, ze vztahu

$$T = 4 * N * M.$$

Počet vrcholů pak může být až $3 * T$. Algoritmus tedy bude paměťově velmi náročný. Implementovány byly dva různé přístupy pro přidělování paměti, které jsou i se svými výhodami a nevýhodami popsány v následujících odstavcích.

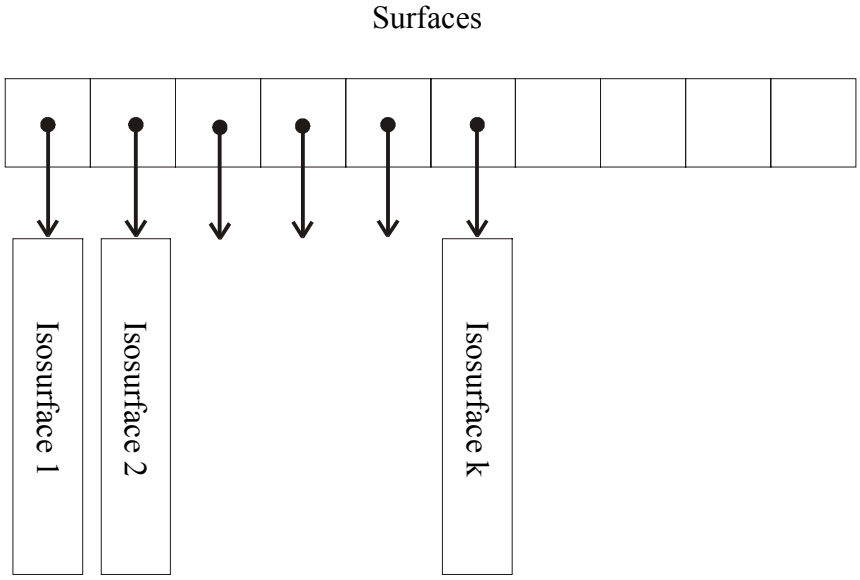
Nárazová alokace

Jak název napovídá, je tento přístup založen na jednorázové alokaci paměti, ke které dochází hned na počátku výpočtu. Program se snaží alokovat paměť pro maximální možný počet trojúhelníků T. Pokud narazí na nedostatek paměti, sníží své požadavky o deset procent (t.j. o hodnotu (T div 10), nikoli o deset procent z požadavku, který je v dané iteraci aktuální) a opět se pokusí alokovat paměť. Pokud se mu to v deseti krocích nepodaří alokovat žádnou paměť a jeho další požadavek by tedy zněl na 0 trojúhelníků, program ohlásí chybu a přeruší výpočet. V opačném případě přikročí

k výpočtu isoploch. Pokud během něho narazí na konec alokované paměti, oznámí uživateli, že některé plochy nemusí být úplné. V opačném případě je přebytečná paměť po dokončení výpočtu uvolněna a struktura předána dalšímu modulu.

Postupná alokace

Tento přístup byl realizován v metodě používající vlákna, a to z toho důvodu, že u předešlého řešení by bylo nutné řešit zápis kritickou sekcí, což by vedlo k výraznému zpomalení výpočtu. Na počátku je pouze alokováno pole ukazatelů na trojúhelníkovou strukturu (Obr.12). Každé vlákno si pak vždy alokuje do struktury, která přísluší právě počítané isoploše, tolik paměti, kolik by odpovídalo maximálnímu počtu trojúhelníků pro jednu isoplochu. Alokace paměti ve vláknech opět probíhá metodou „smlouvání“, kdy vlákno v případě neúspěchu postupně snižuje své požadavky o deset procent původní maximální požadované sumy. Po dokončení výpočtu konkrétní isoplochy je pak přebytečná paměť uvolněna a vlákno přistupuje k výpočtu další plochy. Po doběhnutí všech vláken se pak hlavní program, opět metodou smlouvání, snaží získat paměťový blok, do něhož by již vypočítané plochy zkopíroval. Tentokrát jsou však již známy přesné velikosti jednotlivých ploch, takže se požadavek nesnižuje o konstantní hodnotu, ale vždy o velikost poslední isoplochy v seznamu vygenerovaných povrchů a daná isoplocha se pak před započetím dalšího pokusu o alokaci uvolní z paměti, což zvyšuje pravděpodobnost úspěchu. V takovém případě bude uživatel o této skutečnosti samozřejmě informován a varován, že některé plochy budou ve výstupní struktuře chybět. Tento krok je v některých případech problematický, viz sekce Known issues v příloze B.



Obr.12 Datová struktura pro ukládání isoplch v metodě postupné alokace paměti

Nárazová versus postupná alokace

Přestože se první metoda na první pohled jeví jako „rozhazovačná“, co se paměťových nároků týká, její výhody, jak se nakonec ukázalo, převažují. Výraznou nevýhodou je, že dočasně alokuje velké množství paměti i ve chvíli, kdy výsledným datům bude nakonec stačit malý prostor. Druhá obtíž spočívá v nutnosti využití kritické sekce při paralelním zpracování a následném zpomalení výpočtu.

Výhody oproti postupné alokaci jsou následující:

- Rychlost. Veškerá potřebná paměť se alokuje naráz ještě před započetím výpočtu, na konci nedochází k žádnému kopírování.
- Data jsou po celou dobu v paměti jen jednou. V případě dynamické alokace se v případě nedostatku paměti stává, že se sice spočítají všechny isoplochy, není však už dostatek místa pro alokaci souvislého bloku, do něhož by bylo možné spočtené plochy zkopírovat. Některá data se tak musí smazat a na výstupu bude nakonec méně ploch než v případě nárazové alokace.
- Pokud je potřeba swapovat, musí se při dynamické alokaci často vyměňovat jednotlivé stránky paměti, a to především ve fázi závěrečného kopírování. V případě jednorázové alokace se při výpočtu postupuje od začátku do konce, takže systém musí jen občas prohodit jednu stránku, takže nedochází k žádnému výraznému zpomalení výpočtu.

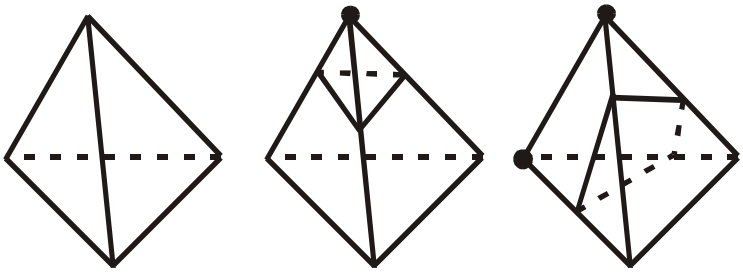
Pozn.: V programu je navíc definována konstanta, která určuje maximální množství paměti, kterou si smí program alokovat. Tato konstanta je v použité kompilaci rovna 900MB.

4.4 Rozbor implementovaných metod

V této části bude nejdříve uveden postup extrakce jednoho plátu isoplochy z jednoho konkrétního tetrahedronu. Dále bude následovat popis implementace metod, které slouží k vyhledání těch čtyřstěnů, které je potřeba během extrakce povrchu odpovídajícího dané hodnotě prozkoumat.

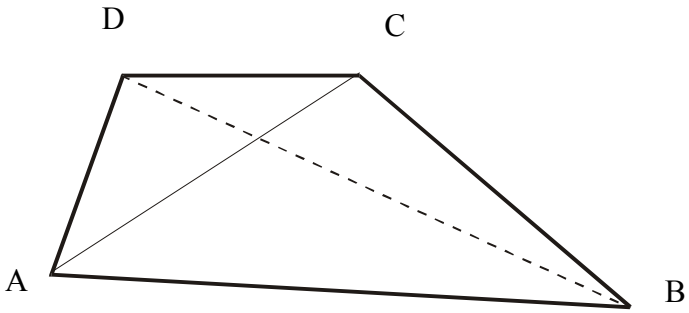
4.4.1 Extrakce plátu isoplochy

V kapitole 3.1.1, v části Marching Tetrahedra, jsou popsány případy protnutí tetrahedronové buňky isoplochou, které mohou nastat (Obr.13).



Obr.13 Možné vztahy mezi tetrahedronem a isoplochou

V prvním případě isoplocha čtyřstěnem neprochází. V druhém vznikne jeden trojúhelník, který je poté bez dalšího zpracovávání přidán k isoploše. Třetí případ však dává vzniknout čtyřúhelníku, který je nuthé rozdělit na dva trojúhelníky. To lze provést dvěma způsoby. V této implementaci byla za kritérium pro rozhodování, kterou ze dvou možných hran využít, vzata délka této hrany. Využita bude tedy vždy kratší z obou možných hran (Obr.14).



Obr.14 V tomto případě je úsečka AC kratší než úsečka BD. Bude tedy vybrána jako nová hrana.

Kromě případů zobrazených na Obr.14 však může nastat i situace, kdy se hodnoty v některých vrcholech budou přímo rovnat hledané isohodnotě. Tato situace je podrobněji rozebrána v odstavci 4.3.1.

4.4.2 Naivní algoritmus

Tento algoritmus nedisponuje žádným aparátem, který by mu umožnil předem rozpoznat, které tetrahedrony mohou být protnuty isoplochou, a kterým se naopak isoplocha úplně vyhne. Tento algoritmus je v podstatě obdobou algoritmu Marching Tetrahedra používaného pro pravidelné tetrahedronové sítě. I tomto případě totiž musí algoritmus navštívit všechny buňky datové množiny, a to pro každou hledanou isohodnotu zvlášť. Pojem „Marching“ v názvu této metody by však mohl být nahrazen pojmem „Leaping“, jelikož v nepravidelné síti nejsou čtyřstěny uspořádány, a proto jimi algoritmus neprochází postupně, ale přeskakuje z jednoho místa na druhé.

Tento algoritmus nevyžaduje žádné předzpracování, samotný proces extrakce je však na druhou stranu velmi pomalý. Algoritmus proto není vhodný pro výpočet většího počtu isoploch. Tato skutečnost byla při implementaci využita. Při extrakci malého počtu ploch není zapotřebí znát normály ve vrcholech všech tetrahedronů tvořících zpracovávaný datový objekt. Stačí vyjádřit pouze normály ve vrcholech těch tetrahedronů, které budou danou plochou, případně plochami, protnuty. Tento krok zpomalí samotný výpočet isoplochy, dává však možnost vyhnout se poměrně časově náročnému preprocessingu.

Aby bylo možné porovnat rychlost jednotlivých algoritmů, umožňuje vytvořený modul využít tuto metodu k extrakci většího počtu isoploch, přestože je tento způsob

velmi neefektivní. Současně je také možné zvolit, jestli budou normály ve vrcholech tetrahedronů počítány či nikoli a pakliže ano, v které fázi.

4.4.3 Swapping Arrays

Tento algoritmus je pouze modifikací naivního algoritmu, jehož implementace je popsána v předchozím odstavci. Jeho použití je výhodné ve chvíli, kdy uživatel požaduje výpočet většího počtu isoploch naráz, přičemž rozsah prahových hodnot, pro něž budou hledány příslušné povrchy, je předem znám. Metoda Swapping Arrays totiž využívá globální koherence mezi jednotlivými isoplochami.

Změna oproti naivnímu algoritmu spočívá v tom, že tato metoda si udržuje dvě pole prvků. Před započetím extrakce první isoplochy jsou do pole nazvaného *Current* uloženy indexy všech buněk vstupního souboru volumetrických dat. Toto pole je pak postupně prohledáno pro první, nejnižší, isohodnotu, přičemž proces průchodu polem se shoduje s naivní metodou. Současně se však indexy všechny čtyřstěnů, jejichž maximální hodnota převyšuje danou isohodnotu, ukládají do pole *Next*. Po nalezení první isoplochy ukazatele na obě pole prohodí, odtud pochází název metody, a k dalšímu zpracování je vybrána isohodnota z opačného konce intervalu požadovaných prahových hodnot, v tomto případě tedy ta nejvyšší. Pro ni je pak opět prohledáno pole current, tentokrát již bez tetrahedronů, které byly vyřazeny při předchozím průchodu. V této fázi jsou do pole next naopak ukládány pouze ty tetrahedrony, jejichž minimální hodnota je nižší, než hodnota hledaná. Po dokončení průchodu se opět prohodí ukazatele na pole current a pole next a cyklus se dále opakuje pro nejnižší, dosud nespočítanou, hodnotu.

Stejně jako u naivní metody je možné zvolit, v které fázi výpočtu, a zda vůbec, budou určovány normály ve vrcholech tetrahedronů. Velikost urychlení tohoto algoritmu závisí jak na geometrickém charakteru vstupních dat, tak i na počtu a rozsahu zadaných isohodnot.

4.4.4 MinMax Lists

Metoda MinMax List byla popsána v podkapitole 3.1.2. V této práci je implementována v modifikované podobě. Jak bylo již v obecném popisu tohoto algoritmu naznačeno,

vkládání a mazání položek takzvaného akčního seznamu vyžaduje určitou režii, která algoritmus zpomaluje. Tato implementace se použití akčního seznamu vyhýbá.

V prvním kroku jsou obdobně jako v algoritmu Gilese a Haimese vytvořeny seznamy s minimálními a maximálními hodnotami všech tetrahedronů (tzv. *minlist* a *maxlist*) a také globálně největší rozdíl Δz, mezi minimální a maximální hodnotou v rámci jedné buňky. Oproti původní metodě je však seřazen pouze seznam minimálních hodnot, a to s využitím algoritmu QuickSort. Tento krok probíhá ve vlákně, a to z toho důvodu, aby bylo v budoucnu v případě potřeby seřazení také maxlistu možné vykonat obě operace paralelně.

Fáze extrakce začíná tím, že jsou v minlistu bisekcí nalezeny indexy poslední buňky, jejíž minimální hodnota je menší nebo rovna hledanému prahu q, a první buňky, jejíž minimální hodnota je větší nebo rovna hodnotě $q - \Delta z$. Interval mezi těmito dvěma hodnotami je poté sekvenčně prohledán. Pro každý zkoumaný tetrahedron je nejdříve ověřeno, zda je jeho maximální hodnota vyšší nebo rovna hodnotě q. Pokud ano, je z tohoto čtyřstěnu vyextrahován plát isoplochy. V opačném případě algoritmus pokračuje další položkou v minlistu.

U této metody není dostupná volba výpočtu normál až během samotného procesu extrakce. Pokud si uživatel přeje, aby byly normály ve vrcholech trojúhelníků počítány, stane se tak již ve fázi předzpracování. Hlavním důvodem je fakt, že metoda byla naimplementována pro paralelní běh ve vláknech. Pokud by tedy měly být normály počítány až v době hledání isoplochy, musela by být tato část programu ohraničena kritickou sekcí, což by výpočet výrazně zpomalilo. Tím spíše proto, že výpočet normál je ve srovnání s nalezením isoploch poměrně pomalou záležitostí.

Druhým důvodem je skutečnost, že tato metoda z principu vyžaduje určitý preprocessing, a proto je vhodné, aby byla použita pro extrakci většího množství isoploch. Tím se zvyšuje pravděpodobnost, že by postupně musely být normály spočítány pro většinu vrcholů tetrahedronové sítě, takže by k úspoře času nakonec nedošlo.

Kapitola 5

Realizace – Vizualizace

V této kapitole bude popsána implementace metod, které slouží k zobrazování neskalárních volumetrických dat.

5.1 Vizualizace vektorových polí

Pro zobrazování vektorových polí byly vybrány dvě metody, a to jednak kontrakce a jednak přímé zobrazování vektorů pomocí úseček.

5.1.1 Kontrakce

Jak již bylo uvedeno v kapitole 3.2, nejedná se o přímé zobrazení vektorového pole. Namísto toho je vybrána jen určitá, v dané situaci zajímavá, vlastnost, čímž je problém převeden do skalární podoby, což umožňuje aplikaci metod pracujících s jednorozměrnými daty v čele s extrakcí isoploch, kterou také tento program využívá. Jako stěžejní vlastnost pro přechod na skalární data byla vybrána velikost vektoru, a to především z toho důvodu, že vektorová pole velmi často obsahují data, která popisují určitý pohyb jako například proudění kapalin či plynů. V takovém případě velikost vektoru odpovídá velikosti rychlosti $|\vec{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$ v daném místě. Zobrazená isoplocha pak ukazuje oblast, v níž je konstantní velikost rychlosti pohybu $|\vec{v}| = const$.

Druhou možností, pro kterou se uživatel může rozhodnout, je uvažovat pouze jednu vybranou složku vektoru. Tak lze například zjistit, v jaké oblasti je konstantní jedna ze složek v_x, v_y nebo v_z vektoru rychlosti.

5.1.2 Přímé zobrazování vektorů

Metoda přímého zobrazování vektorů je vhodným doplňkem kontrakce. Pokud totiž zobrazená isoplocha ukazuje oblast, v níž je například velikost rychlosti konstantní, vektory ukáží směr pohybu v různých místech. Problematická je situace, kdy jsou vektory rozloženy v prostoru příliš hustě, což činí výslednou scénu nepřehlednou. Z tohoto důvodu byly místo šipek použity pouze úsečky, což navíc urychluje zobrazení a zlepšuje možnost manipulace se zobrazenou množinou isoploch. Vektory lze navíc také vynásobit reálným koeficientem, takže je možné délku a směr úseček podle potřeby regulovat. Současně lze ještě zobrazit pole hodnot v podobě bodů. Jejich barva je odstínována podle velikosti skalární hodnoty, z níž je počítána isoplocha, a pozice bodů odpovídá pozici vrcholů buněk volumetrických dat. Je-li tedy nutné určit orientaci vektorů, což representace pomocí úseček nedovoluje, je možné zapnout zobrazení pole hodnot. Každý vektor tak bude vycházet z jednoho barevného bodu, který bude indikovat počátek úsečky.

Kapitola 6

Dosažené výsledky

Všechny implementované algoritmy byly testovány na různě velkých vstupních datech. Hlavním sledovaným kritériem byla rychlost. Výsledky těchto testů jsou shrnuty v následujících odstavcích. V tabulkách nejsou uváděna celá jména metod, místo nich jsou používány zkratky **N** – Naivní metoda, **SA** – Swapping Arrays, **MM(S)** – MinMax Lists – Single Threaded, **MM(P)** – MinMax Lists – Parallel. Všechna měření byla prováděna na počítači s následujícími parametry:

- dva procesory Intel Pentium III 500MHz
- 1GB RAM
- dva pevné disky 6.1GB a 8.6GB
- operačním systémem WinNT4.0ws eng, SP6a

6.1 Závislost doby výpočtu normál na objemu dat

V tabulce 1 jsou uvedeny časy, které jsou potřeba pro výpočet normál ve všech vrcholech původních volumetrických dat.

Počet vrcholů (Počet buněk)	2 500 (15 850)	5 000 (33 000)	10 000 (66 265)	20 000 (133 433)
Naměřený čas [s]	2,75	4,9	8,1	14,6

Tab. 1: Závislost doby výpočtu normál na objemu dat

6.2 Porovnání metod

V následujících třech tabulkách budou pro srovnání uvedeny časy, za které jednotlivé metody vygenerovaly 100 izoploch v rozsahu prahových hodnot od 0 do 100% a s různým počtem vstupních tetrahedronů. Porovnáme-li jednotlivé tabulky Tab. 2 – Tab. 4, zjistíme, že časy v první tabulce jsou nejrychlejší. Ani v jednom z případů nedochází během extrakce k výpočtu normál ve vrcholech tetrahedronů. Rozdíl v hodnotách je způsoben tím, že v prvním případě nedochází ani k interpolaci hodnot z vrcholů tetrahedronů do vrcholů trojúhelníků.

Z tabulek je zřejmé, že nejpomalejší je podle očekávání metoda Naivní. Metoda Swapping Arrays vede jen asi k dvacetiprocentnímu urychlení, a to pouze v případě většího počtu isoploch. Algoritmus MinMax Lists naopak dosahuje až osmdesátiprocentní úspory času. Pro korektní porovnání je samozřejmě zapotřebí uvažovat verzi MM(S), která běží pouze v jednom vlákně.

6.2.1 Bez výpočtu normál

Během tohoto testu nebyly vůbec počítány normály ve vrcholech původní tetrahedronové sítě.

Počet vrcholů (Počet buněk)	2 500 (15 850)	5 000 (33 000)	10 000 (66 300)	20 000 (133 400)
čas N [s]	5,13	9,54	22,7	37,8
čas SA [s]	4,49	8,44	18,9	30,3
čas MM(S) [s]	1,82	3,3	5,86	10,1
čas MM(P) [s]	1,67	2,24	3,8	5,8

Tab. 2: Porovnání metod – normály nepočítány

6.2.2 S výpočtem normál během předzpracování

Při tomto měření byly normály ve vrcholech tetrahedrnů vypočteny během preprocessingu a v uvedených časech tedy není doba jejich zjištění zahrnuta. Je v nich však zahrnula doba, kterou vyžaduje interpolace těchto hodnot do vrcholů výsledné trojúhelníkové sítě.

Počet vrcholů (Počet buněk)	2 500 (15 850)	5 000 (33 000)	10 000 (66 300)	20 000 (133 400)
čas N [s]	7,8	13,33	25,4	50
čas SA [s]	6,31	10,42	21,63	39,2
čas MM(S) [s]	4,06	4,7	7,5	12,48
čas MM(P) [s]	2,82	3,2	4,75	8,37

Tab. 3: Porovnání metod – normály vypočítány předem

6.2.3 S výpočtem normál v průběhu extrakce

V tomto testu byla měřena rychlost výpočtu s tím, že normály byly počítány až v průběhu samotné extrakce. Toto měření nemohlo být provedeno s metodou MinMax List. Vzhledem k tomu, že tento algoritmus byl implementován pro paralelní běh, je při jeho použití umožněn výpočet normál pouze v době preprocessingu. (Podrobněji viz podkapitola 4.4.4).

Počet vrcholů (Počet buněk)	2 500 (15 850)	5 000 (33 000)	10 000 (66 300)	20 000 (133 400)
čas N [s]	8,9	16,4	31,6	61,2
čas SA [s]	7,01	13,23	28,6	47,4

Tab. 4: Porovnání metod – normály počítány během extrakce

6.3 Předzpracování

Tento test byl zaměřen na to, abychom mohli určit, zda je vždy vhodné vypočítávat normálové vektory ve vrcholech původní tetrahedronové sítě předem, v rámci preprocessingu. Výsledky jasně prokázaly, že pro malý počet isoploch je vhodnější ponechat výpočet normál až do fáze samotného procesu extrakce, jelikož v takovém případě jsou zpracovávány jen ty vrcholy, které jsou skutečně potřeba. Hranici, od které je již výhodné využít předzpracování však nelze přesně určit, neboť závisí na hledané isohodnotě.

Vrcholů: 20 000	Extrahována 1 plocha		Extrahováno 10 ploch	
Metoda	N	SA	N	SA
S předzpracování	17,5 (16,9 + 0,6)	15,3 (14,7 + 0,6)	22,7 (16,8 + 5,9)	17,1 (14,3 + 2,8)
Bez předzpracování	1,9 (0 + 1,9)	2,6 (0 + 2,6)	10,7 (0 + 10,7)	7 (0 + 7)

Tab. 5: Výhodnost předzpracování

Kapitola 7

Závěr

Všechny použité algoritmy extrahují isoplochy pro zadané prahové hodnoty korektně. Vytvořený renderer poskytuje prostředky pro manipulaci a zkoumání zobrazené informace.

Testy ukázaly, že zatímco metoda Swapping Arrays jen mírně urychluje takzvanou Naivní metodu, algoritmus MinMax Lists, který využívá odlišného přístupu, dosahuje znatelně lepších výsledků. Je-li navíc spuštěn na počítači se dvěma procesory, doba pro výpočet se oproti jednoprocesorovému stroji sníží zhruba o jednu třetinu až o jednu polovinu.

Jako nedostatečná se ukázala kvalita současného algoritmu pro odhad normál ve vrcholech tetrahedronů původní sítě s využitím pouze informace obsažené ve vstupních datech a nezohledňující tvar výsledné isoplochy.

Byl navržen a rozpracován nový přístup, který uvažuje i možnost rovnosti hodnot ve vrcholech vstupní tetrahedronové sítě a zadané prahové hodnoty.

7.1 Další práce

- Zdokonalení algoritmu pro výpočet normál isoplochy
- Doplnění rendereru o další funkce tak, aby měl uživatel ještě lepší možnost k důkladnému prozkoumání zobrazených objektů. Jedná se například o

možnost nastavení rychlosti posunu a rotace objektu, ovládání z klávesnice a podobně.

Literatura a odkazy

[Cign96] Cignoni, P., Montani, C., Puppo, E., Scoigno, R.: *Optimal isosurface extraction from irregular volume data.* Proceedings of the 1996 Symposium of Volume Visualisation, San Francisco, CA USA, 1996, str. 31-38.

[Gile90] Giles, M., Haimes, R.: *Advanced interactive visualization for cfd.* Computing Systems in Engineering, 1(1):51-62, 1990.

[Hans93] Hansen, Ch.: *Visualization of Vector Fields (2D and 3D).* Tutorial, Los Alamos National Laboratory, 1993.

[Shen95] Shen, H., Johnson, C. R.: *Sweeping simplicies: A fast iso-surface extraction algorithm for unstructured grids.* Proceedings of Visualisation '95, IEEE Computer Society Press, Los Alamitos, CA., 1995.

[Sper90] Speray, D., Dennon, S.: *Volume Probes: Interactive Data Exploration on Arbitrary Grids.* Computer Graphics, 1990, 24(5):1-12.

[Watt92] Watt, A., Watt, M.: *Advanced Animation and Randering Techniques.* Addison-Wesley, New York, 1992.

[Wilh92] Wilhelms, J., Van Gelder, A.: *Octrees for faster isosurface generation.* ACM Transactions on Graphics, 11(3):201-227, July 1992.

[Zara98] Žára, J., Beneš, B., Felkel, P.: *Moderní počítačová grafika.* Computer Press, Praha, 1998.

[WWW1] *Visualization of Vector Fields*, http://amira.zib.de/usersguide/tutvector.html

[WWW2] *Visualizing Local Properties and Characteristic Structures of Dynamical Systems*, http://www.cg.tuwien.ac.at/~helwig/diss/node3.htm

[WWW3] *Flow Visualization Techniques*, http://www.cs.wpi.edu/~matt/courses/cs563/talks/flowvis/flowvis.html

[WWW4] *Visualization of ground water flow and transport using LIC*, http://www.gkw-gmbh.de/grundw/paper/lic/lic.htm

PŘÍLOHA A

User Guide

Introduction

This document serves as a user guide for the MVE modules *IsosurfaceExtractor* and *IsosurfaceRenderer*, which can be found in the *Isosurface.dll* library. In the following paragraphs you will learn what purpose do these modules serve for as well as how to use them within the Modular Visualization Environment.

IsosurfaceExtractor - Module Function

Briefly put, the *IsosurfaceExtractor* module was designed to extract isosurfaces from an irregular tetrahedra mesh. Therefore, the input of the *IsosurfaceExtractor* module is the Tetrahedra 2 MVE type. As the result of its computations, the module offers a set of surfaces represented by triangle meshes, so the Surfaces MVE type is the module's output type. These surfaces may then be displayed in the *Isosurfacerenderer* module or the whole set of triangle meshes may also be saved into a file in the STL format.

IsosurfaceExtractor - Using the module within the MVE

Since the *IsosurfaceExtractor* module requires a tetrahedra represented object with multiple values in each vertex, it may only follow modules that produce a set of tetrahedra in the Tetrahedra 2 data structure. That is, for instance, the *TetraGeneratorMulti* module. As described above, the module itself produces a set of surfaces, thus being determined to precede modules that accept the Surfaces data structure as their input. That is, at this moment, only the *IsosurfaceRenderer* module.

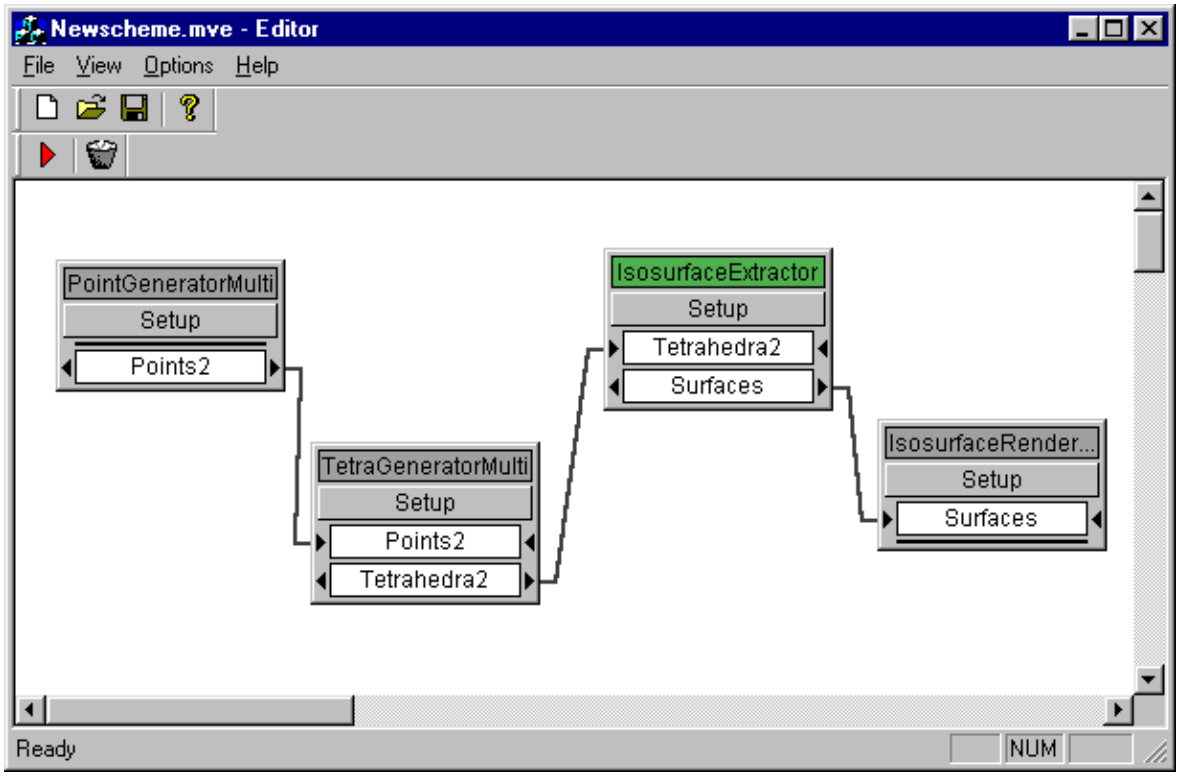


Figure 1: An example of the usage of the IsosurfaceExtractor *module.*

IsosurfaceExtracor - setup

It is possible to adjust the *IsosurfaceExtractor* module parameters by clicking the Setup button on the module icon on the MVE main screen before the whole diagram is executed. Although the range of acceptable values can not be calculated at the moment, users have a chance to insert the desired values, so that the whole diagram of modules can be run smoothly with no interruption. The value may be entered either directly (in case that the user is familiar with the data he/she works with) or in percentage. The setup window is displayed on the following picture.

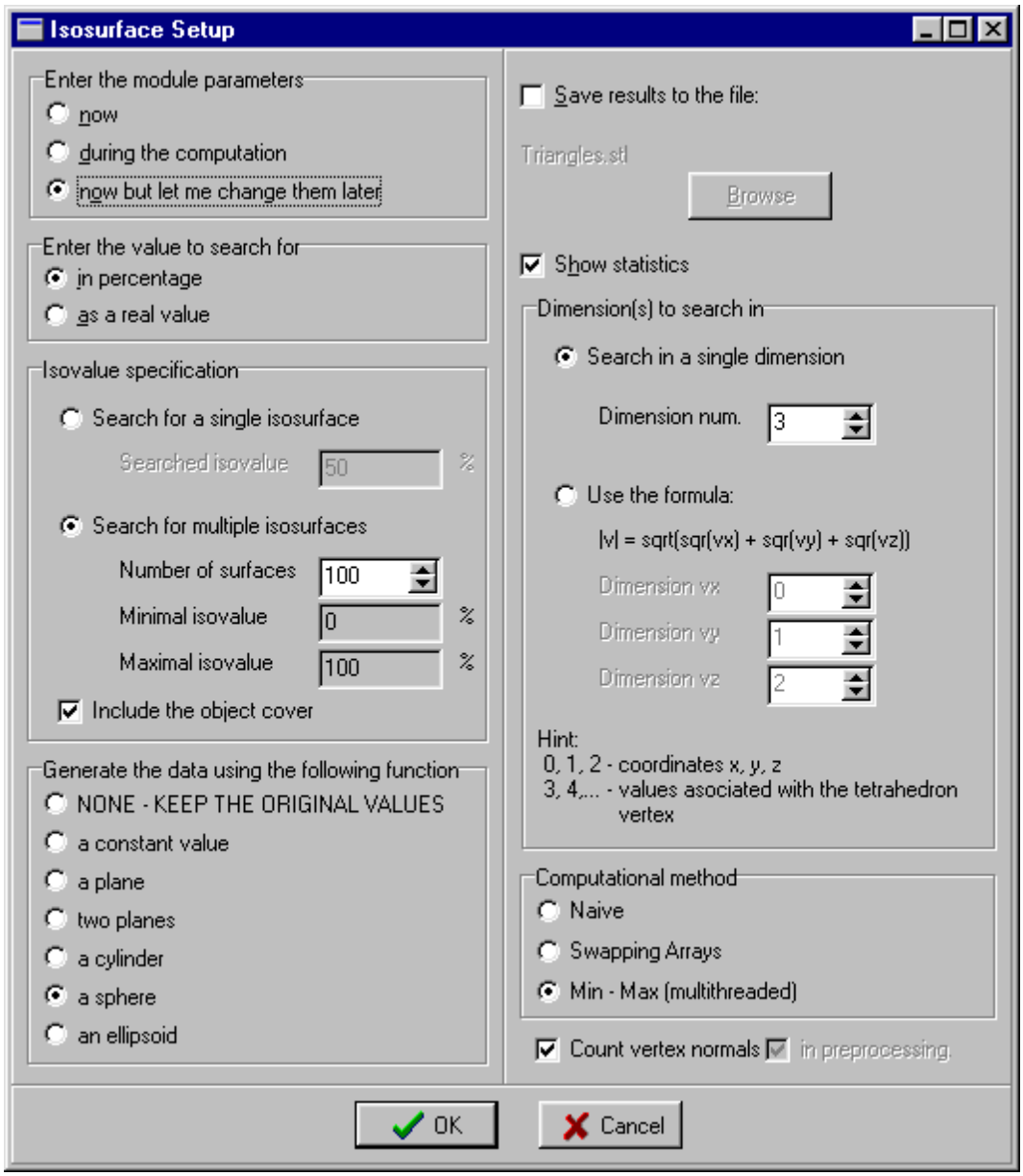


Figure 2: The Setup window of the IsosurfaceExtractor *module*

In the *Enter the module parameters* box the user can choose whether to setup the module right at the moment, during the computation or whether to set it up at the moment yet keeping a chance to change it later. Having checked the appropriate option, the user must choose, whether he/she will enter the isovalue in percentage or directly as a real value.

For the description of the remaining options on the Setup form, see the Setup window description section below.

If the user decides not to setup the module in advance using the Setup window (i.e. other option than *now* is checked in the first box), a similar setup window will appear when the *IsosurfaceExtractor* module gets focus from the MVE. By that time, *IsosurfaceExtractor* has already obtained the input data, calculated the minimal and maximal values that can be searched for and displayed these on the form.

Setup window description:

- Isovalue specification - This is the value or range of values that the *IsosurfaceExtractor* module will search for. The values must fit the <Minimal value, Maximal value> interval, otherwise an error message appears. Also, in case of multiple surfaces search, the Minimal isovalue must be lower then the Maximal isovalue.

- Constant value - Pressing this button attaches the value *one* to all the vertices. Consequent searching will then result in displaying the whole object.

- Plane - Pressing this button makes the module consider the x coordinate of each vertex as the value to extract isosurfaces from. Consequent searching will then result in displaying a plane.

- Two Planes - Pressing this button makes the module consider the absolute value of the x coordinate of each vertex as the value to extract isosurfaces from. Consequent searching may then result in displaying one or two planes, according to the location of the original object.

- Cylinder - Pressing this button makes the module consider the distance from the *y* axis of each vertex as the value to extract isosurfaces from. Consequent searching will then result in displaying a part

of a cylinder.

- Sphere - Pressing this button makes the module consider the distance from the center point S [mid_x,mid_y,mid_z] of each vertex as the value to extract isosurfaces from. Consequent searching will then result in displaying a sphere.

- Ellipsoid - Pressing this button makes the module consider the distance from the center point S [mid_x,mid_y,mid_z] of each vertex, where the *y* coordinate is divided by 2, as the value to extract isosurfaces from. Consequent searching will then result in displaying a part of an ellipsoid.

- Save results to file: - Saves the computed set of surfaces into a file using the STL format. The file path and the name may be specified by clicking the Browse button.

- Show statistics - If this option is checked, the module will display a Statistics window after the computation has finished.

- Dimension(s) to search - Specifies whether the module should search for the surface in just one dimension or if it should use the formula for the vector length.

- Computational method - Specifies the method to be used for the extraction.

- Count normals - Specifies, whether the normal vectors should be computed and if so, then if it should be done during the preprocessing phase. When the MinMax List method is chosen, the vectors can only be computed in preprecessing.

IsosurfaceRenderer - Module Function

The *IsosurfaceRenderer* module serves for displaying the data contained in the Surfaces MVE data structure, which is the input of the module. This involves displaying multiple isosurfaces, either shaded or in the wire structure mode, visualizing the surfaces' normal vectors, animating through the sequence of the isosurfaces or sets of normal vectors and displaying the value and vector fields. It also offers various settings to adjust the colors and the length of all the vectors being displayed to the current needs. The *IsosurfaceRenderer*'s main window resembles the following figure. All the above described features can be reached easily using the window's main menu.

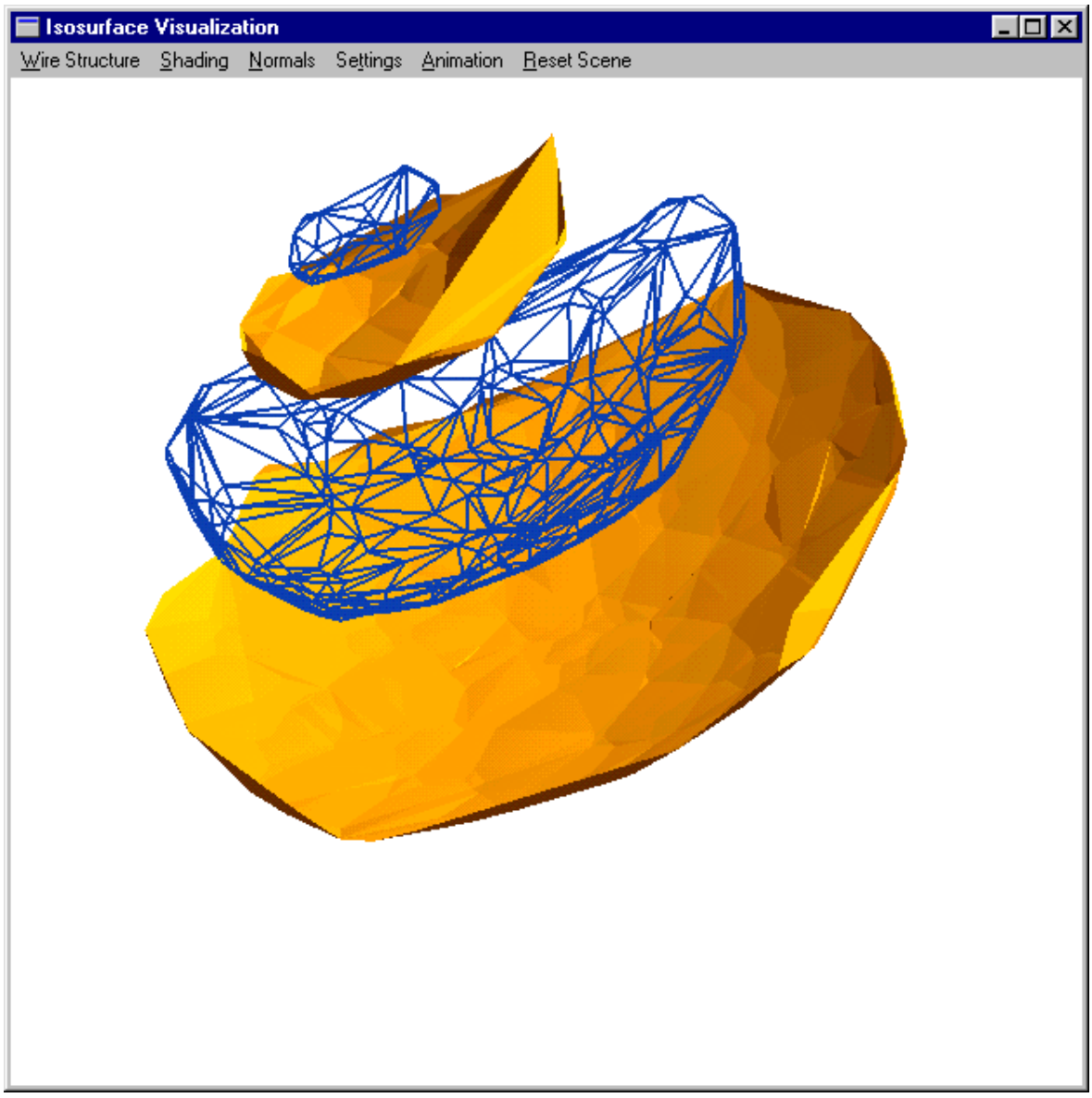


Figure 3: A screenshot of the IsosurfaceRenderer *main window*

PŘÍLOHA B

Installation Guide

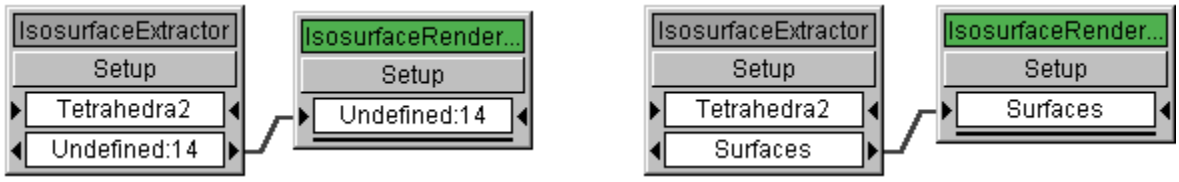
The *IsosurfaceExtractor* and the *IsosurfaceRenderer* modules are both stored within the *Isosurface.dll* library. The modules do not require an installation different than any other MVE modules. See the MVE documentation for details.

Data type label

Only in the case that there is a label *Undefined : 14* on the modules' icons instead of the *Surfaces* type (see picture bellow), it is necessary to add the following line to the [MVE path]/Cfg/Types.def file:

Surfaces 14

The reason is that the Surfaces type is quite new so in the older versions of the MVE_Editor, this type might not be registered. This will, however, not reduce the functionality of the modules.



Wrong! Ok!

Requirements

The *IsosurfaceRenderer* module requires the OpenGL library to be available on the computer.

Known issues

The only problematic point is, that during a simultaneous computation of a big number of isosurfaces using the MinMax List method, which was designed for the parallel run, there are difficulties with the allocation of a larger memory block that is meant to be used as the space for the output data structure (the allocation strategy is described in 4.3.3). While the running threads get as much memory as they require, the main thread (after all the extraction threads finish) does not obtain enough space to copy the results computed by the extraction threads into one block, which could than be passed to other modules. The system, in such a case, reports enough available memory (e.g. more than 1GB) but does not allow to allocate even a fraction of that amount (200MB). The reason is unknown to the author. The *IsosurfaceExtractor* in such a case attempts to allocate less and less, until it gets some memory. Then it copies at least those surfaces, for which there is enough space, and informs the user about the situation.

This problem, however, appeared only on the Windows NT system with the core for the multiprocessors. On other systems, which only support the pseudo parallel computations, these troubles were not encountered during the testing phase.

PŘÍLOHA C

Programmer‘s Guide

The Program Structure

The program is divided into ten files, which may be classified according to their usage into the following three groups:

• General Files	• Isosurface.dpr • Setup.pas • MyTypes.pas	These are the files that are used generally in both modules.
• IsosurfaceExtracor Files	• CountIso.pas • SortThread.pas • ExtractThread.pas • Statistics.pas	These are the files that are used by the IsosurfaceExtractor module.
• IsosurfaceRenderer Files	• ShowIso.pas • ColorSettings.pas • WireSettings.pas • FieldSettings.pas	These are the files that are by the IsosurfaceRenderer module.

General Files

The *Isosurface.dpr* file is the main file of the library. It provides and exports the functions that are required for the communication and data transfers between the modules and the MVE program.

The *Setup.pas* file serves only for managing the Setup dialog and checking the values, set by the user, for validity before copying them into the MySetupData structure, in which they are later passed to the IsosurfaceExtractor module.

In the *MyTypes.pas* file all the types used throughout the project are defined.

IsosurfaceExtractor files

The *CountIso.pas* is the file that is responsible for the functionality of the IsosurfaceExtractor module. If the MinMax Lists method is chosen, it cooperates with the *SortThread.pas* and the *ExtractThread.pas* files. The *Statistics.pas* file only displays the information about the computation, if this information is required.

IsosurfaceRenderer files

The ShowIso.pas is the main file of the IsosurfaceRenderer module. It is responsible for all the visualization functions. For its work, it utilizes the ColorSettings.pas, WireSettins.pas and the FieldSettings.pas forms, which provide the ShowIso form with the information about the settings adjusted by the user.

-