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

**Diplomová práce**

# Metody triangulace v paralelním prostředí

Plzeň, 2013 Michal Šmolík

# Zadání

Západočeská univerzita v Plzni
Faculty of Applied Sciences
Akademický rok: 2012/2013

Studijní program: Computer Science and Engineering
Forma: Full-time
Obor/komb.: Počítačová grafika a výpočetní systémy (PGS)

**Podklad pro zadání DIPLOMOVÉ práce studenta**

| PŘEDKLÁDÁ: | ADRESA | OSOBNÍ ČÍSLO |
|---|---|---|
| Bc. ŠMOLÍK Michal | Baarova 821, Rokycany - Plzeňské Předměstí | A11N0147P |

**TÉMA ČESKY:**

Metody triangulace v paralelním prostředí

**NÁZEV ANGLICKY:**

Triangulation methods in a parallel environment

**VEDOUCÍ PRÁCE:**

Prof. Ing. Václav Skala, CSc. - KIV

**ZÁSADY PRO VYPRACOVÁNÍ:**

1) Seznamte se s metodami triangulace v E^2 a E^3.
2) Porovnejte výpočetní složitost a paměťové nároky.
3) Realizujte daný algoritmus v paralelním prostředí.
4) Vlastnosti algoritmu prokažte na datech velkého rozsahu.
5) Dosažené výsledky porovnejte s referenčními algoritmy.

**SEZNAM DOPORUČENÉ LITERATURY:**

Student si provede odbornou rešerši sám.

**Podpis studenta:** ..... **Datum:** .....

**Podpis vedoucího práce:** ..... **Datum:** .....

# Prohlášení

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

V Plzni dne …… ……

Michal Šmolík

# Poděkování

Děkuji vedoucímu této diplomové práce prof. Ing Václavu Skalovi, CSc., za hodnotné rady a odborné vedení během vzniku této diplomové práce. Diplomová práce ověřující metody triangulace a tetrahedronizace v paralelním prostředí na GPU a CPU vznikla na základě algoritmů navržených prof. Skalou. Základní postupy v $E^2$ byly ověřeny v rámci semestrální práce, na které jsem spolupracoval s kolegou Bc. Lukášem Karlíčkem (viz [Šmo12]).

# Anotace

Triangulace a tetrahedronizace jsou velmi rozšířené problémy nejen v počítačové grafice. Cílem této práce je navrhnout novou metodu pro triangulaci bodů v $E^2$ a v $E^3$. Navrhnutá metoda je realizovatelná pomocí paralelního výpočtu, díky čemuž se sníží časová náročnost celé triangulace.

# Abstract

Triangulation and tetrahedronization are widespread problems not only in computer graphics. The aim of this thesis is to propose a new method for triangulation of points in $E^2$ and in $E^3$. The proposed method is realizable using parallel computation and thus the time required for triangulation is reduced.

# Obsah

# 1 Úvod

Triangulace je velmi známý problém ve výpočetní geometrii. Využívá se ve velkém množství aplikací, jakými je například robotika, počítačové vidění ( = computer vision), syntéza obrazů a samozřejmě také v matematických a přírodních vědách. Ve 2D je jedná o tvorbu trojúhelníkové sítě nazývanou triangulace, ve 3D se jedná o tvorbu tetrahedronové sítě nazývanou tetrahedronizace (lze využívat i pojmu triangulace).

Příklad použití triangulace je možné ukázat na častém případě v počítačové grafice. Uvažujme množinu bodů ve dvourozměrném prostoru, kde v každém bodě máme naměřenou hodnotu, která definuje výšku. Takto zadaná množina bodů nám definuje nějaký terén, který chceme vizualizovat. V takovém případě nastupuje triangulace, která převede množinu bodů na trojúhelníky. Poté lze výsledné trojúhelníky zobrazit, a tím dostaneme spojitý terén z množiny disjunktních bodů. Navíc je možné na trojúhelníky aplikovat texturu a získat tak daleko věrohodnější zobrazení terénu nebo jednoduše pomocí lineární interpolace vypočítat hodnotu výšky i v bodech, které nejsou v množině definující terén [Fuk06]. Jinou možností dopočtu neznámých hodnot výšky je využití metody radiálních bázových funkcí ( = RBF) [SkV12]. Navíc je samozřejmě možné, že hodnota v zadaných bodech neurčuje výšku, ale může definovat např. teplotu nebo tlak. V daných bodech mohou být naměřeny nejen skalární, ale také vektorové hodnoty, jako je směr proudění vody v určitém bodě. Naším cílem je poté vhodně prezentovat tyto diskrétní data. V nejjednodušším případě je možné zobrazit pouze tyto data, ovšem většinou je nutné odvodit i hodnoty v bodech, kde měření neproběhlo. Tím se opět dostáváme na triangulaci a tetrahedronizaci, která pomocí trojúhelníků a tetrahedronů dokáže vhodně prezentovat hodnoty i v místech, kde nebylo měřeno [Koh02].

Jelikož je triangulace velice rozšířená ve výše uvedených oborech, bylo vytvořeno mnoho algoritmů, jak ji konstruovat. Nejvíce algoritmů bylo vymyšleno pro nejznámější a nejpoužívanější Delaunayovu triangulaci, která při vhodné implementaci dává dobrou časovou složitost a také nejlepší výsledky, co se tvaru vytvořené sítě týče. Pokud je Delaunayova triangulace implementována optimálně, její časová složitost dosahuje v očekávaném případě $O(N \log N)$, ovšem pro velké množství dat je zpracování pomalé. Zlepšení a zrychlení algoritmu pro triangulaci může být dosáhnuto pomocí optimalizačních technik, které mohou snížit složitost základního algoritmu. Další možností je využití paralelního přístupu při triangulaci. Hlavním problémem paralelních algoritmů triangulace je typicky jejich mnohem větší komplikovanost a náročnější implementace než u sériových verzí triangulací.

## 1.1 Hlavní cíle

Cílem této diplomové práce je navrhnout, implementovat a otestovat novou paralelní metodu triangulace a tetrahedronizace. Hlavní myšlenka, která bude v této

práci využita je následující. Pokud se v neuspořádané množině bodů v $E^2$ nebo v $E^3$ zavede částečné uspořádání, je toho možné využít pro jednodušší a rychlejší triangulaci. Pomocí částečného uspořádání je možné vstupní body rozdělit do několika nezávislých množin a ty samostatně triangulizovat. Jednotlivé triangulace se poté spojí, aby se získala výsledná triangulace vstupní množiny bodů.

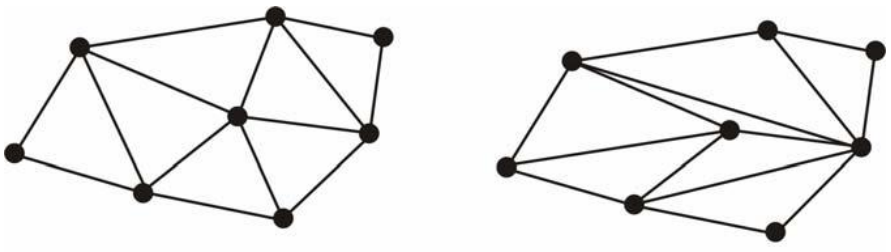
## 1.2 Použité prostředky

Testování triangulace a tetrahedronizace probíhalo na školním počítači, jehož sestava je popsána níže:

- CPU: Intel(R) Core(TM) i7 920 (4 × 2,67GHz) s 8 HyperThreads
- paměť: 12 GB RAM
- GPU: 2 × GeForce GTX 295
    - 30 multiprocesorů × 8 CUDA Cores na jeden multiprocesor (1,38GHz) (každé GPU)
    - paměť: 896MB (1,05GHz) (každé GPU)
- operační systém: Microsoft Windows 7 64bit

# 2 Triangulace v $E^2$

Triangulace je proces převodu množiny bodů na trojúhelníkovou síť. Je zřejmé, že pro každou množinu bodů existuje více než jedna trojúhelníková síť (viz Obr. 2.1).



Obr. 2.1: Obrázek ukazuje dvě možné varianty propojení bodů do trojúhelníkové sítě (převzato z [Fuk06]).

Obecně lze triangulaci definovat takto [Bay]:

*Triangulace $T$ nad množinou bodů $P$ představuje takové planární rozdělení, které vytvoří soubor m trojúhelníků $t = \{t_1, t_2, \ldots, t_m\}$ a hran tak, aby platilo:*

- *Libovolné 2 trojúhelníky $t_i, t_j \in T, (i \neq j)$, mají společnou nejvýše jednu hranu.*
- *Sjednocení všech trojúhelníků $t \in T$ tvoří $CH(P)$.*
- *Uvnitř žádného trojúhelníku neleží žádný další bod z $P$.*

Z Eulerovy věty lze odvodit vztah mezi počtem bodů $n$, počtem hran $n_h$ a počtem trojúhelníků $n_t$ v rovině pro triangulaci $T$, pokud $k$ bodů leží na $CH(P)$ [Bay]:

$$\begin{aligned} n_h &= 3n - 3 - k, \\ n_t &= 2n - 2 - k. \end{aligned} \tag{2.1}$$

Vztah (2.1) lze dále zjednodušit tak, že bude pouze funkcí $n$:

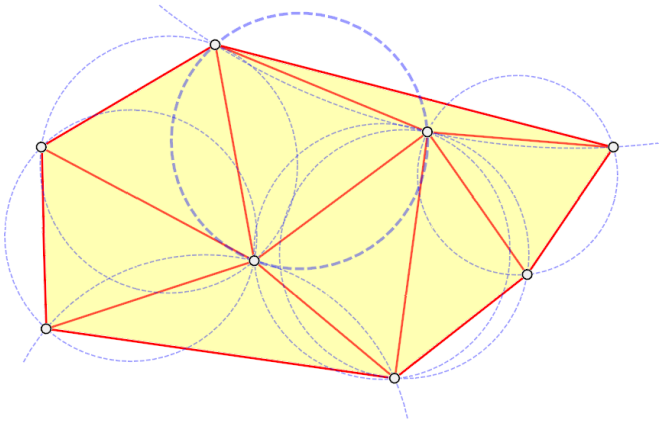
$$\begin{aligned} n_h &\leq 3n - 6, \\ n_t &< 2n - 5. \end{aligned} \tag{2.2}$$

Tyto vztahy platí obecně pro jakoukoliv planární triangulaci a je možné je využít k odhadu počtu trojúhelníků nebo hran v triangulaci.

## 2.1 Delaunayova triangulace

Delaunayova triangulace je vůbec nejrozšířenější triangulací, která se v praxi používá. Její hlavní výhoda spočívá ve vlastnostech, které nabízí. Její nejdůležitější vlastností je to, že vytváří trojúhelníky, které se nejvíce blíží trojúhelníkům

rovnostranným, což je při vykreslování požadováno. Tenké a dlouhé trojúhelníky totiž při vykreslování dávají špatné výsledky, neboť je nutné, aby se barva interpolovala na úzké ploše, a vždy je zvýhodněn jeden z vrcholů konkrétního trojúhelníku. Další špatnou vlastností je numerická nestabilita. Z těchto důvodů je požadováno, aby trojúhelníky byly co nejvíce rovnostranné (viz Obr. 2.2).



Obr. 2.2: Příklad Delaunayovy triangulace (převzato z [Bay]).

Mezi nejzákladnější vlastnosti Delaunayovy triangulace patří [Bay]:

- Uvnitř kružnice $k$ opsané libovolnému trojúhelníku $t_j \in DT$ neleží žádný jiný bod množiny $P$.
- $DT$ maximalizuje minimální úhel $\alpha$ $\forall t$, avšak $DT$ neminimalizuje maximální úhel $\alpha$ v $t$.
- $DT$ je lokálně i globálně optimální vůči kritériu minimálního úhlu.
- $DT$ je jednoznačně určena, pokud žádné čtyři body neleží na kružnici.

Tyto vlastnosti zajišťují, že Delaunayova triangulace dává vždy největší minimální úhel mezi všemi triangulacemi. To znamená, že tato triangulace nám dává nejrovnostrannější trojúhelníky, ze všech známých metod. Tato vlastnost je docílena prohazováním hran, které musejí splňovat následující podmínku [Fuk06]:

*Nechť* ***ab*** *je hrana triangulace* $\Delta = (V, E)$ *společná dvěma trojúhelníkům* ***abc*** *a* ***adb***. *Hrana* ***ab*** *je lokálně optimální podle Delaunayova kritéria, pokud opsaná kružnice trojúhelníku* ***abc*** *neobsahuje protější bod* ***d*** *a zároveň pokud opsaná kružnice trojúhelníku* ***adb*** *neobsahuje protější bod* ***c***.

Obecně může být hrana lokálně optimální i podle jiného kritéria, ovšem v Delaunayovské triangulaci je lokální optimalita docílena prohazováním hran, což plně odpovídá výše uvedeným vlastnostem Delaunayovy triangulace.

Při určování, zda je konkrétní hrana lokálně optimální, je nutné zjistit, zda leží protější bod sousedícího trojúhelníku uvnitř kružnice opsané, která je definována body daného trojúhelníku. První možností, jak toho docílit je vypočítat střed a poloměr

kružnice. Z těchto parametrů poté určit rovnici kružnice a dosazením protějšího bodu testovat, zda bod leží uvnitř nebo vně. Vztah odpovídající této možnosti má následující tvar [Fuk06]:

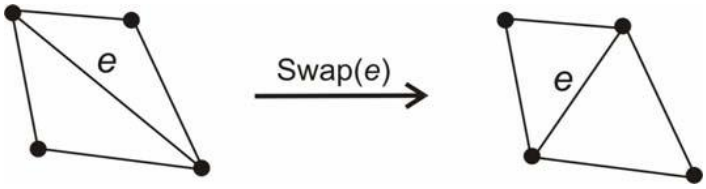
$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0, \tag{2.3}$$

kde $x, y$ jsou souřadnice testovaného bodu, $x_0, y_0$ je střed kružnice a $r$ poloměr. Pokud je nerovnost splněna, testovaný bod se nachází uvnitř kružnice a hrana tedy není lokálně optimální. Tento výpočet je ovšem zbytečně složitý. Existuje snazší varianta, kdy není nutné hledat koeficienty rovnice kružnice, ale vystačíme si pouze se zadanými body. K otestování, zda bod leží uvnitř nebo vně kružnice opsané, nám stačí vyčíslit hodnotu determinantu

$$det = \begin{vmatrix} a_x & a_y & a_x^2 + a_y^2 & 1 \\ b_x & b_y & b_x^2 + b_y^2 & 1 \\ c_x & c_y & c_x^2 + c_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{vmatrix}, \tag{2.4}$$

kde $a, b, c$ jsou body definující kružnici opsanou a $d$ je testovaný bod. Pokud je $det > 0$, leží bod uvnitř kružnice. V opačném případě tj. $det < 0$, leží vně. Pokud je $det = 0$ leží testovaný bod na kružnici. Při výpočtu determinantu je nutné dávat pozor na orientaci bodů, neboť v případě opačného zadání pořadí bodů je nutné výsledky invertovat. V tomto případě jsou výsledky platné pro body, které jsou orientované proti směru hodinových ručiček [Fuk06].

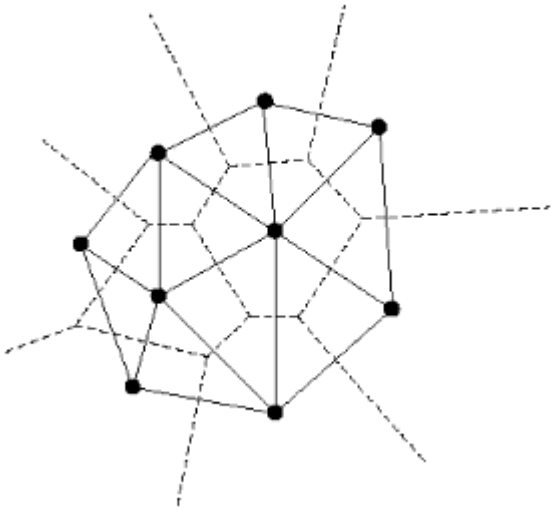
Pokud testem zjistíme, že hrana není lokálně optimální, je nutné tuto hranu prohodit, aby splňovala Delaunayovo kritérium. Po jejím prohození již nemusíme otestovat tutéž hranu sousedícího trojúhelníku, neboť bude podmínku automaticky splňovat. Příklad prohození hrany (anglicky $flip$ nebo $swap$) je vidět na (Obr. 2.3).




Obr. 2.3:Příklad prohození hrany (převzato z [Fuk06]).

Jelikož je Delaunayova triangulace nejrozšířenější, vzniklo pro ni mnoho algoritmů, jak ji sestrojit. Každý z těchto algoritmů má své výhody i nevýhody. Obecně lze tyto algoritmy rozdělit na dvě skupiny – metody přímé a nepřímé. Jednotlivé přímé metody jsou uvedeny v dalších kapitolách. Do nepřímých metod patří konstrukce Delaunayovy triangulace přes Voronoiův diagram. Je totiž dokázáno, že Delaunayova triangulace je duální Voronoiovu diagramu. To znamená, že je možné nejprve sestrojit Voronoiův diagram, který lze převést na Delaunayovu triangulaci (viz Obr. 2.4). Tento

postup se v praxi nepoužívá, neboť sestrojení Voronoiova diagramu je složitější než samotná Delaunayova triangulace.



Obr. 2.4: Příklad Voronoiova diagramu a k němu duální Delaunayova triangulace (převzato z [Koh02]).

### 2.1.1 Lokální prohazování

Tento algoritmus je založen na principu prohazování hran. Dokud nejsou všechny hrany triangulace lokálně optimální, jsou prohazovány. To znamená, že vstupem musí být již nějaká hotová triangulace, kterou poté převádíme na Delaunayovu. Pseudokód tohoto algoritmu by mohl vypadat následovně [Fuk06]:

*Vytvoř jakoukoliv triangulaci* $T(V, E)$ *nad vstupní množinou vrcholů* $V$*;*
***begin***
    ***While*** *(T není lokálně optimální)* ***do***
    ***begin***
        ***If*** *(existuje lokálně neoptimální hrana* $e$*)* ***then***
            *Prohoď*$(e)$*;*
    ***end****;*
***end****.*

Algoritmus tedy pracuje tím způsobem, že nejprve vytvoří nějakou triangulaci. Poté vloží všechny hrany do zásobníku obsahující hrany, které nejsou lokálně optimální. Postupně je ze zásobníku vybírá a legalizuje je. To, že některá hrana je již lokálně optimální neznamená, že se ve výsledné triangulaci vyskytne. Je možné, že prohozením jiné hrany konkrétního trojúhelníku se jeho tvar změní, a tím se hrana stane opět lokálně neoptimální. Proto je nutné po prohození jedné hrany, vložit ostatní hrany trojúhelníku do zásobníku. Algoritmus končí, když je zásobník prázdný [Fuk06].

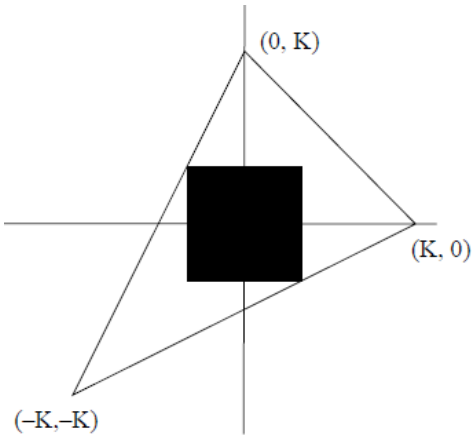
Výhodou tohoto algoritmu je snadná implementace a velmi dobrá stabilita. I přes jeho výhody se však tento algoritmus v praxi téměř nepoužívá, neboť časová složitost dosahuje v nejhorším případě $O(N^2)$ a pro průměrný případ $O(N)$. Další nevýhoda je

v nutnosti předem zkonstruovat počáteční triangulaci. Paměťová náročnost algoritmu je lineární nebo-li $O(N)$.

### 2.1.2 Inkrementální vkládání

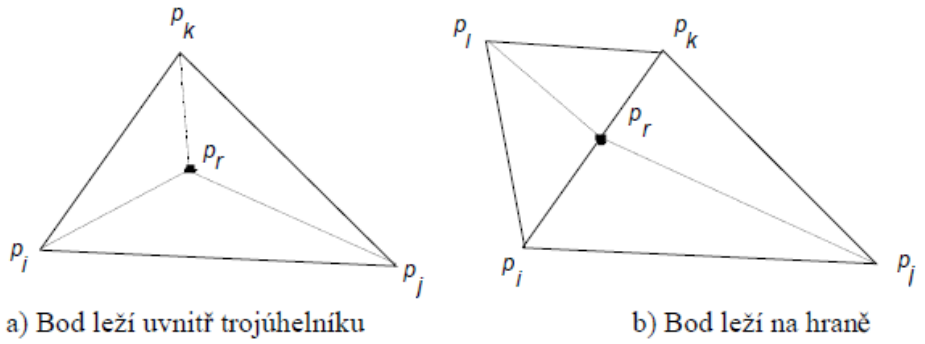
Algoritmus inkrementálního vkládání je asi nejvíce používaný pro tvorbu Delaunayovy triangulace. Jeho největší výhoda spočívá v jednoduchosti, robustnosti a snadné rozšiřitelnosti do 3D. Dále je možné tento algoritmus snadno upravit pro triangulaci s povinnými hranami. Složitost algoritmu dosahuje $O(N^2)$ v nejhorším případě a při vhodné implementaci $O(N \log N)$ v očekávaném případě [Koh02]. Paměťová náročnost algoritmu je lineární nebo-li $O(N)$.

Jak je možné vyčíst z názvu algoritmu, je založen na principu postupného vkládání bodů do hotové Delaunayovy triangulace. Vstupem je množina bodů, které budou postupně přidávány. První krok algoritmu spočívá v nalezení tzv. počátečního simplexu. To znamená, že se snažíme vytvořit počáteční trojúhelník, který bude ohraničovat všechny body vstupní množiny. Velikost tohoto trojúhelníku je většinou odvozena z $min - max$ boxu vstupní množiny (viz Obr. 2.5).




Obr. 2.5: Volba počátečního simplexu (převzato z [Koh02]).

Další krok algoritmu je postupné vkládání bodů vstupní množiny do aktuální Delaunayovy triangulace. Vybereme ze vstupní množiny jeden bod a hledáme, ve kterém trojúhelníku se nachází. Po nalezení tohoto trojúhelníku je nutné bod do triangulace zařadit tím, že původní trojúhelník vyhodíme a nahradíme ho novými trojúhelníky. Zde mohou nastat dvě situace - bod leží uvnitř trojúhelníku nebo bod leží na hraně trojúhelníku. V prvním případě nahradíme původní trojúhelník trojicí nových trojúhelníků. V případě druhém musíme hranu rozdělit a trojúhelníky obsahující tuto hranu nahradit dvojicí nových trojúhelníků (viz Obr. 2.6).




Obr. 2.6: Vložení nového bodu do triangulace (převzato z [Koh02]).

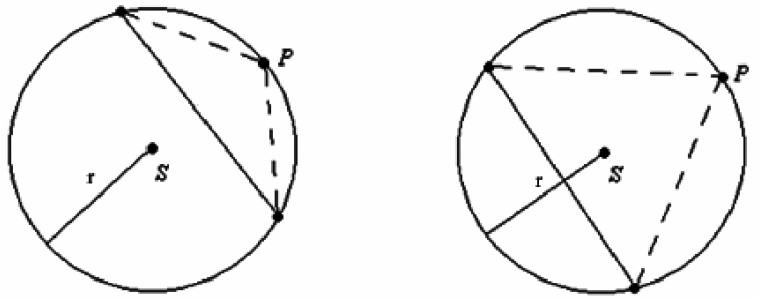
Po vložení nového bodu do Delaunayovy triangulace je nutné ověřit, zda je nově vzniklá triangulace Delaunayovská. Je tedy potřeba otestovat, zda nově vložené trojúhelníky mají lokálně optimální hrany (viz kap. 2.1.1). Pokud nově přidané trojúhelníky nejsou lokálně optimální, je potřeba prohodit jednu z jejich hran. Tato operace ovšem může vyvolat to, že sousední trojúhelník se stane lokálně neoptimální a je třeba ho legalizovat. Tímto způsobem je možné v nejhorším případě projít všechny trojúhelníky.

Po vložení všech bodů vstupní množiny do triangulace pouze stačí vyhledat trojúhelníky, které nepatří do výsledné triangulace. To jsou takové trojúhelníky, které obsahují alespoň jeden vrchol původního simplexu. Po odstranění těchto trojúhelníků získáme výslednou Delaunayovu triangulaci.

### 2.1.3 Inkrementální konstrukce

Algoritmus inkrementální konstrukce je velmi jednoduchý a je založen na vlastnosti Delaunayho triangulace nejmenší kružnice opsané. Tento algoritmus pracuje následujícím způsobem. Na počátku je zvolen libovolný bod množiny určené k triangulaci. K tomuto bodu je nalezen nejbližší bod (tj. bod, který má nejmenší Euklidovskou vzdálenost). Po sestrojení první hrany triangulace z těchto dvou nalezených bodů se vyhledá další bod napravo od této hrany, který s danou hranou tvoří trojúhelník a má nejmenší Delaunayho vzdálenost. Ta je definován následujícím způsobem [Lei06].

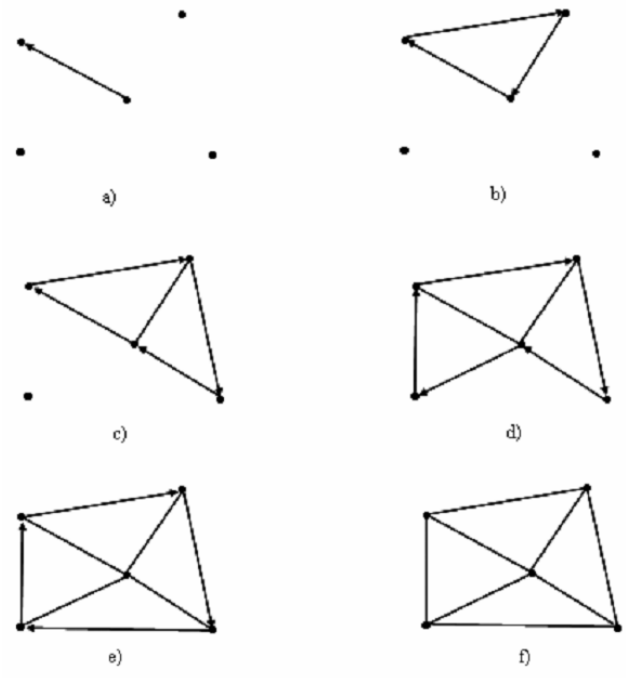
Pokud střed kružnice opsané hraně $e$ a bodu $P$ leží ve stejné polorovině jako bod $P$ potom je Delaunayho vzdálenost rovna $r$ jinak $-r$, kde $r$ je poloměr kružnice opsané. Poloroviny jsou dány právě hranou $e$, tak jak je vidět na Obr. 2.7. Pokud bod $P$ nebyl nalezen, otočí se orientace hrany a vyhledávání bodu se opakuje, tentokrát nalevo od hrany $e$. Po nalezení bodu $P$ je vytvořen první trojúhelník ze tří hran. Je důležité tyto hrany udržovat s jednotnou orientací. Tímto je dokončena inizializace algoritmu.




Obr. 2.7: Vlevo případ výpočtu, kdy S neleží ve stejné polorovině jako bod P, vpravo případ, kdy bod S leží ve stejné polorovině jako P. (převzato z [Lei06]).

Nyní jsou hrany přidány do AEL (Active Edge List = Seznam aktivních hran). Vezme se první hrana z tohoto seznamu a vyhledá se pro ni bod *P* s nejmenší Delaunayho vzdáleností. Pokud je nalezen, vytvoří se dvě nové hrany a přidají se do AEL. Přidání se ovšem provádí pouze za předpokladu, že přidávaná hrana není již v AEL obsažena. V tomto případě je z AEL odstraněna a přidána do výsledné triangulace. Aktuální hrana, pro kterou byl bod hledán je odstraněna z AEL a přidána do výsledné triangulace. Pokud pro aktuální hranu nebyl nalezen žádný bod, znamená to, že hrana je součástí konvexní obálky a je proto přidána také do výsledné triangulace [Lei06].

Celý postup končí ve chvíli, kdy je AEL prázdný. Díky udržování orientace hran v jednom směru na hranici aktuální triangulace uložené v AEL se testuje vždy hrana spolu s body, které jsou napravo (resp. nalevo) od orientovaných hran. Zjednodušeně tento postup vystihuje (Obr. 2.8), na kterém jsou hrany hranice aktuální triangulace znázorněny jako orientované a hrany výsledné triangulace jsou neorientované.




Obr. 2.8: Principiální znázornění konstrukce DT pomocí inkrementální konstrukce (převzato z [Lei06]).

Složitost algoritmu inkrementální konstrukce je v nejhorším případě $O(N^2)$, použitím vhodných datových struktur lze získat očekávanou složitost $O(N \log N)$. Ve zvláštních případech při pravidelném rozložení vstupní množiny bodů lze získat očekávanou složitost $O(N)$. Paměťová náročnost algoritmů inkrementální konstrukce je složitosti $O(N)$.

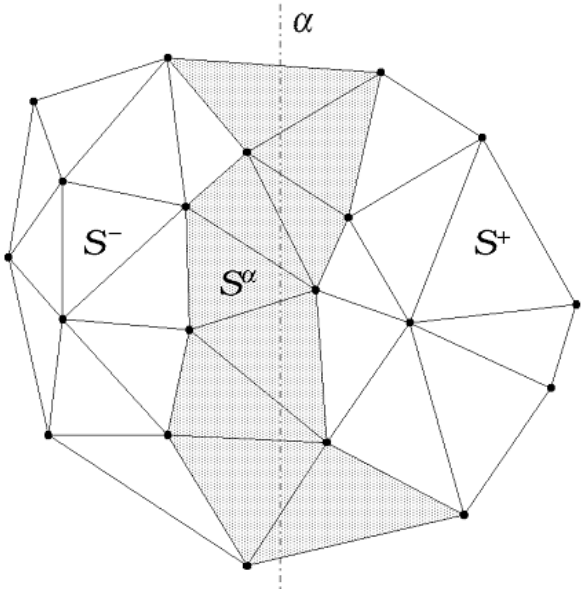
### 2.1.4 Paralelní metody

Se zvyšujícím se počtem bodů pro triangulaci je čím dál výhodnější využít nějaký algoritmus konstrukce triangulace, který je možné provádět paralelně. Díky paralelizmu lze často získat urychlení oproti sériové verzi triangulace. Obecně jsou ovšem paralelní metody náročnější na implementaci. Několik metod pro tvorbu triangulace bude popsáno v následujících kapitolách.

#### 2.1.4.1 De-Wall

Algoritmus paralelní triangulace nazývaný De-Wall [Cig93][Cig98] je založen na technice rozděl a panuj. Vstupní množina bodů $P$ může být jednoduše rozdělena s využitím dělící roviny (přímky) $\alpha$ na dvě části $P^-$ a $P^+$. Největší složitost algoritmu je ve spojovací části dvou triangulací $S^-$ a $S^+$ vytvořených z bodů $P^-$ a $P^+$.

Dělící rovina (přímka) $\alpha$ rozdělí prostor $E^2$ na dvě části $\alpha^+$ a $\alpha^-$. Zároveň $\alpha$ rozdělí triangulaci na tři části. Trojúhelníky, které jsou protnuty pomocí $\alpha$, jsou označeny $S^\alpha$, trojúhelníky obsažené pouze v části $\alpha^+$, jsou značeny $S^+$, a trojúhelníky obsažené pouze v části $\alpha^-$, jsou značeny $S^-$ (viz Obr. 2.9).




Obr. 2.9: Rozdělení triangulací na tři části pomocí dělící přímky $\alpha$ (převzato z [Cig93]).

$S^\alpha$ je množina trojúhelníků, které mají být vytvořeny ve fázi spojování triangulací $S^-$ a $S^+$ a zároveň platí, že pro každý trojúhelník $s \in DT(P)$, $s$ je protnut pomocí $\alpha$ pokud $s \in S^\alpha$.

Algoritmus paralelní triangulace De-Wall lze popsat pomocí následujících kroků algoritmu.

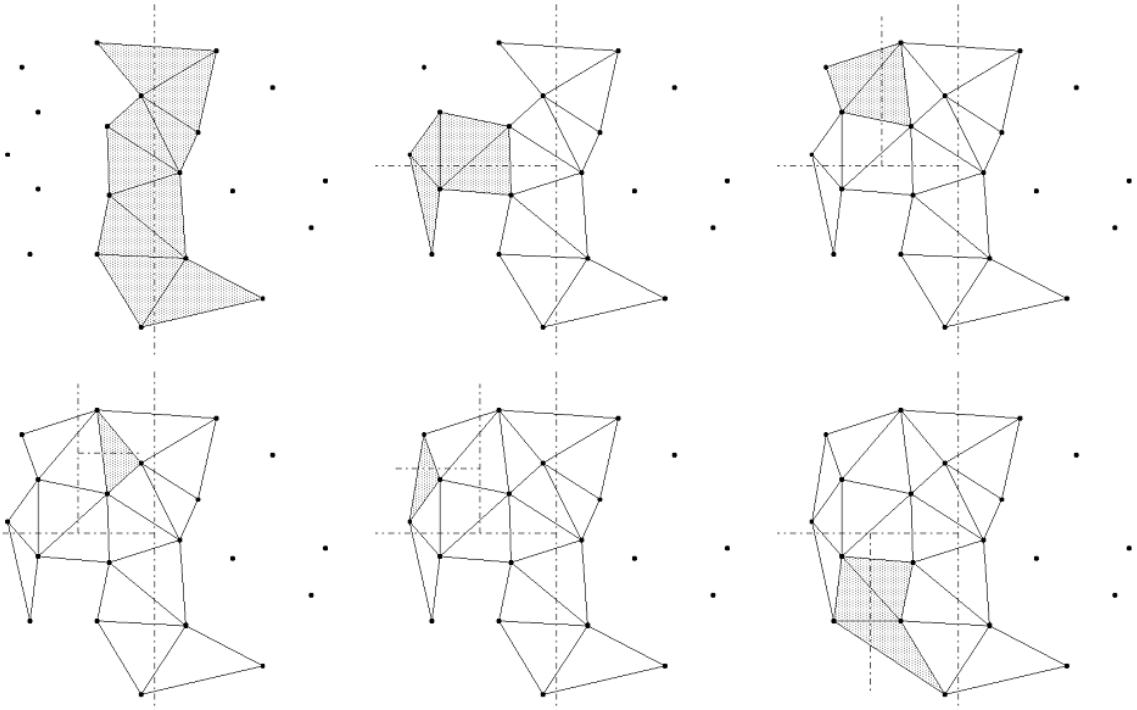
- Vybrání dělící přímky $\alpha$.
- Konstrukce $S^\alpha$ a dvou množin bodů $P^-$ a $P^+$.
- Rekurzivně použít De-Wall na množinu bodů $P^-$. Začít od $S^\alpha$ a vytvořit $S^-$.
- Rekurzivně použít De-Wall na množinu bodů $P^+$. Začít od $S^\alpha$ a vytvořit $S^+$.
- Vrátit všechny trojúhelníky z $S^\alpha$, $S^-$ a $S^+$.

Spojovací část triangulace $S^\alpha$ může být vytvořena pomocí algoritmu inkrementální konstrukce. Algoritmus je zahájen od trojúhelníku $s \in DT(P)$ a inkrementálně vytváří $DT(P)$ pomocí přidávání nového trojúhelníku v každém kroku bez nutnosti modifikovat aktuální triangulaci. Při vytvoření nového trojúhelníku jsou nové hrany vloženy do struktury „Active Face List“ ($AFL$) a v případě, že hrana je v $AFL$ již obsažena, pak je z $AFL$ odstraněna a není znovu vkládána. Strukturu $AFL$ je nutné rozdělit na tři následující skupiny.

- $AFL^\alpha$: hrana je protnuta dělící přímkou $\alpha$.
- $AFL^+$: hrana je celá obsažena v $\alpha^+$.
- $AFL^-$: hrana je celá obsažena v $\alpha^-$.

$S^\alpha$ je vytvořeno inkrementálně pomocí $AFL^\alpha$ (viz Obr. 2.10). Vráceny jsou trojúhelníky $S^\alpha$ a struktury $AFL^+$ a $AFL^-$. V dalším kroku je De-Wall rekurzivně

aplikován na dvojici ($P^-$, $AFL^-$) a ($P^+$, $AFL^+$). Dělící přímka $\alpha$ je cyklicky vybírána jako rovnoběžka s osou $x$, $y$.



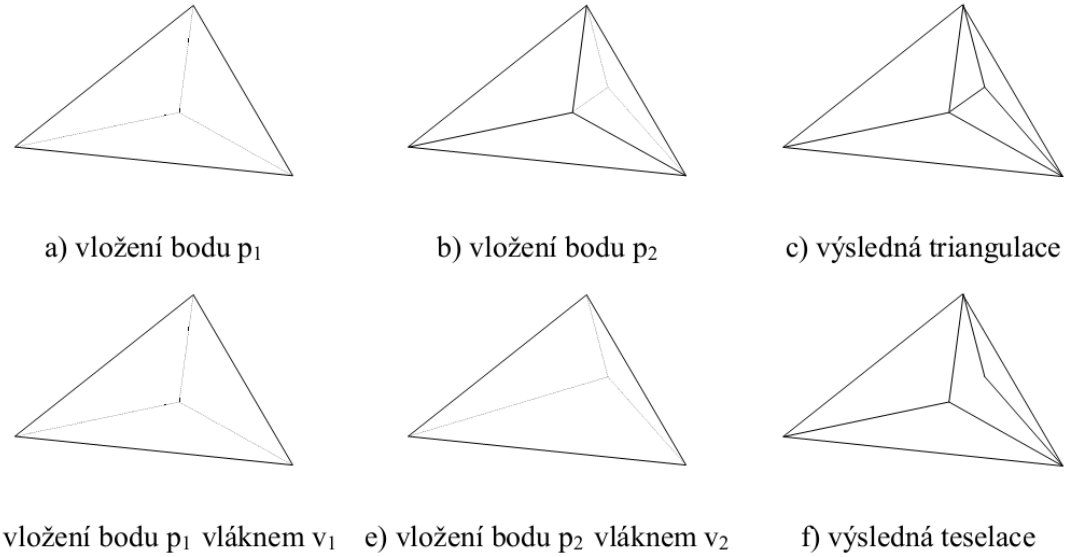
Obr. 2.10: Dělení bodů a vytváření triangulací $S^\alpha$ (převzato z [Cig93]).

Volba úplně prvního trojúhelníku obsaženého v první množině $S^\alpha$ je provedena následujícím způsobem. Je nalezen bod $p_1 \in P$ nejbližší k dělící přímce $\alpha$. Poté je nalezen druhý bod $p_2 \in P$ takový, že $p_2$ leží na druhé straně $\alpha$ než $p_1$ a zároveň $p_2$ je bod s minimální euklidovskou vzdáleností od $p_1$. V dalším kroku je hledán bod $p_3$, pro který je poloměr opsané kružnice trojúhelníku ($p_1$, $p_2$, $p_3$) minimální.

Časová náročnost algoritmu je v nejhorším případě $O(N^2)$, ale může být dosaženo očekávané složitosti $O(N \log N)$. Paměťová náročnost algoritmu je $O(N)$ [Cig98].

#### 2.1.4.2 Modifikace inkrementálního vkládání

Paralelní algoritmus pro konstrukci Delaunayovy triangulace, který bude popsán v této kapitole, je založen na inkrementálním vkládání a byl popsán v [Koh02] [Koh05]. Sériový algoritmus triangulace nelze beze změny spustit paralelně a očekávat správný výsledek. Na (Obr. 2.11) je znázorněn případ, kdy došlo k vytvoření neplatné triangulace. Je tedy nutné zajistit nějaký způsob synchronizace, aby modifikace téhož trojúhelníku neprovádělo více vláken současně.




Obr. 2.11: Porovnání sériového vkládání bodů (a, b, c) a paralelního vkládání bodů (d, e, f) do triangulace (převzato z [Koh02]).

Paralelizaci Delaunayovy triangulace lze provést pomocí tří různých přístupů a to dávkový, pesimistický a optimistický. Dávkový přístup je založen na principu, že několik vláken bude současně vyhledávat trojúhelníky a pouze jedno vlákno bude vkládat nové body do nalezených trojúhelníků a provádět legalizaci. Tímto přístupem je zabráněno možným konfliktům, kdy více vláken upravuje stejný trojúhelník. Nicméně při větším počtu paralelních vláken nebude vlákno, které vkládá body, stíhat a výsledné urychlení paralelní verze nebude příliš značné.

Dalším z uvedených přístupů je pesimistický přístup. Všechna vlákna vykonávají stejný kód. Je zde ovšem rozdíl ve vyhledávání trojúhelníků, které mohou provádět všechna vlákna paralelně, a v modifikaci trojúhelníků, kterou je možné provádět vždy pouze pomocí jediného vlákna a je nutné k tomu využívat synchronizačního primitiva.

Při využití optimistického přístupu se využívá předpokladu, že vložení jednoho bodu do triangulace změní triangulaci pouze lokálně. Ve skutečnosti ovšem legalizace může změnit až celou triangulaci. Vyhledávání trojúhelníků je možné provádět jako při pesimistickém přístupu. Rozdíl je při vkládání a legalizaci triangulace. Při této fázi se vždy zamknou všechny trojúhelníky, které se budou měnit. V praxi jsou body vkládány vlákny na různá místa v triangulaci, a tudíž vlákna nebudou muset často čekat na odemčení trojúhelníků. Popis algoritmu pracovního vlákna u optimistického přístupu paralelní triangulace je popsán níže.

***begin***

***for*** *r:=0* ***to*** *$N_k$-1* ***do begin*** *// vlož $p_r$ do DT(S)*

*Nalezni v DAG simplex obsahující $p_r$;*

***while*** *dělený simplex nebo jeho sousedé jsou uzamčeni někým jiným* ***do***

*čekej;*

*Uzamkni simplex a všechny jeho sousedy;*

*Rozděl simplex a uzamkni nové simplexy;*

***while*** *legalizované simplexy nebo jejich sousedé jsou uzamčeni někým jiným* ***do***

*čekej;*

*Uzamkni legalizované simplexy a jejich sousedy;*

*Legalizace;*

*Odemkni všechny simplexy uzamčené tímto vláknem;*

*Vzbuď všechna vlákna, která čekala na dokončení tohoto vlákna;*

***end****;*

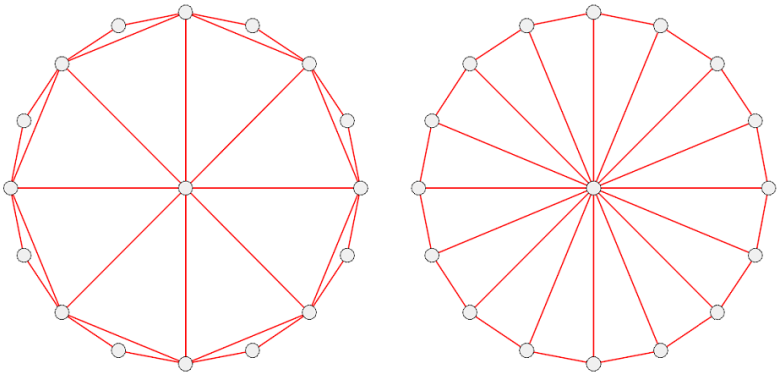
***end****;*

Body je nutné rozdělit do několika disjunktních množin, aby každé vlákno mělo své body, které bude vkládat do triangulace. Množinu bodů je možné buďto rozdělit staticky na stejně velké části nebo druhý způsob je množinu bodů nějakým způsobem uspořádat a poté až rozdělit. Pro uspořádání je možné využít seřazení bodů podle $x$ nebo $y$ souřadnice nebo podle vzdálenosti od počátku. Samotné řazení bodů má výslednou složitost $O(N \log N)$. Ve výsledcích bylo zjištěno, že časová náročnost triangulace je nižší, pokud byly vstupní body v počátku uspořádány (rozděleny podle geometrické polohy). Používané uspořádání bylo buďto podle $x$-ové, nebo $y$-ové souřadnice. Výsledná očekávaná časová složitost paralelní triangulace je $O(N \log N)$ a paměťová náročnost $O(N)$.

## 2.2 Greedy triangulace

Triangulace na množině bodů $V \epsilon R^2$ je Greedy triangulací s následujícími vlastnostmi [Záb05]:

- Za předpokladu, že žádné dvě hrany nemají stejnou délku, je Greedy triangulace jednoznačná.
- Z hlediska úhlových kritérií nedává Greedy triangulace příliš dobré výsledky, protože tvar a úhly trojúhelníku se při konstrukci sítě neberou v potaz (viz Obr. 2.12).
- Greedy triangulace bývá používána jako základ heuristik pro MWT (= minimum weight triangulation, triangulace s minimální váhou) – minimální váhy sice obecně nedosahuje, ale je z hlediska váhového kritéria lokálně minimální.
- Změny bodů GT mají globální vliv.
- Začlenění povinných hran je snadné.



Obr. 2.12: Srovnání Greedy (vlevo) a Delaunaovy (vpravo) triangulace (převzato z [Bay]).

Hltavý (Greedy) algoritmus je takový, který se nikdy nevrací a nemění nic v již hotové části triangulace. V každém kroku je do triangulace přidána jedna hrana a proces tvorby skončí po dosažení celkového počtu hran triangulace. Celkový počet hran je dán počtem bodů a počtem hran konvexní obálky. Základní algoritmus žravé triangulace je implementačně jednoduchý. Problémem je vysoká časová složitost v nejhorším případě $O(N^3)$ a paměťová náročnost $O(N^2)$ [Záb05].

Algoritmus probíhá v těchto krocích:

- Vygeneruje všechny možné hrany mezi body množiny $V$ a uloží je do seznamu hran $E$.
- Vzestupně seřadí seznam hran $E$ podle délky.
- Vybere nejkratší hranu z dosud nezpracovaných hran a přijme ji do triangulace $GT(V)$, pokud neprotíná žádnou z hran již přijatých.
- Opakuje se předchozí krok, dokud nezbývá žádná hrana.

Hlavním problémem všech algoritmů na konstrukci Greedy triangulace nad nepravidelně rozloženým bodovým polem je jejich vysoká časová složitost. U nejefektivnějších algoritmů bylo dosaženo časové složitosti v nejhorším případě $O(N^2 \log N)$ [Záb05].

## 2.3 Triangulace s omezením

Triangulace s omezením jsou označovány jako Constrained triangulations. Do triangulace jsou zavedeny povinné hrany spojující definované body $p$. Poloha povinných hran se poté při triangulaci již nesmí měnit, ale zároveň je vstupní předpoklad, aby se povinné hrany neprotínaly [Bay].

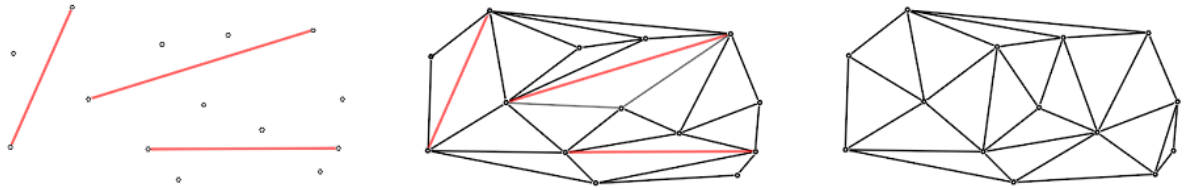
Povinné hrany při konstrukci triangulace kříží jiné možné hrany, které jsou vzhledem k nějakému kritériu lokálně optimální (a pro triangulaci vhodnější), avšak tyto hrany nejsou použity. Triangulace s omezením (se vstupní podmínkou) proto nejsou lokálně optimální.

Široké použití této triangulace je v kartografii při tvorbě digitálních modelů terénu. Povinné hrany umožňují lepší modelování morfologie terénu.

Triangulizovat se vstupní podmínkou lze pomocí Greedy algoritmu (Constrained Greedy triangulation) nebo pomocí Delaunayovy triangulace (Constrained Delaunay triangulation).

### 2.3.1 Greedy triangulace s omezením

Upravením běžné Greedy triangulace lze dosáhnout triangulace s omezením. Tvorba Greedy triangulace s omezením probíhá ve dvou krocích (viz Obr. 2.13). Nejprve jsou do prázdné triangulace přidány povinné hrany, které nemohou být v budoucnu nahrazeny žádnou jinou hranou. Ve druhém kroku probíhá již běžná Greedy triangulace.



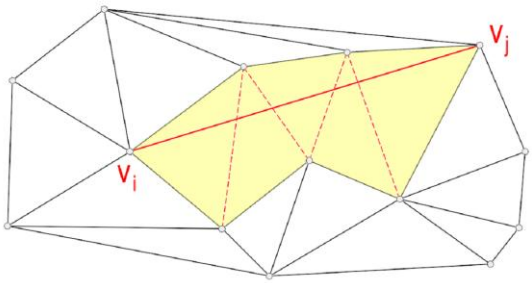
Obr. 2.13: Postup Greedy triangulace s omezením (vlevo) a běžná Greedy triangulace (vpravo) (převzato z [Bay]).

### 2.3.2 Delaunay triangulace s omezením

Delaunayova triangulace s omezením je nejpoužívanější triangulace v geoinformatice. Na rozdíl od běžné Delaunayovy triangulace je nutná redefinice, že přes povinné hrany neprobíhá swapování. Triangulace jako celek nemaximalizuje minimální úhel v trojúhelnících. Zavedením povinných hran je snížena rychlost triangulačního algoritmu.

Konstrukce triangulace probíhá ve třech krocích. Nejprve se vytvoří běžná Delaunayova triangulace, poté jsou do vytvořené triangulace zadány povinné hrany. Jako poslední krok je nutné provést převod triangulace s vloženými povinnými hranami na triangulaci s omezením. Převod probíhá v několika krocích:

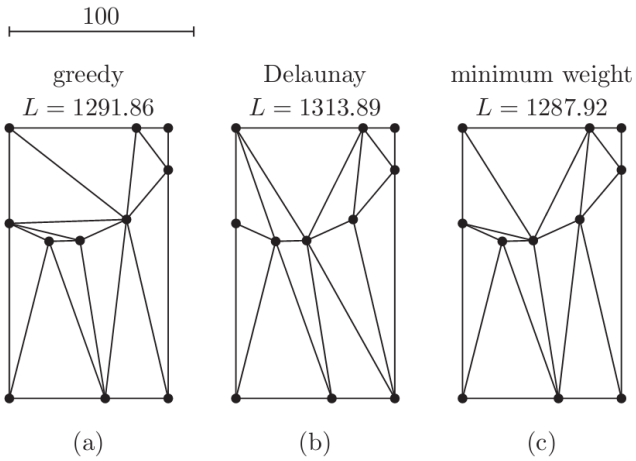
- Nalezení stran Delaunayovy triangulace protínajících povinnou hranu (viz Obr. 2.14).
- Zrušení všech stran v Delaunayově triangulaci protínajících povinnou hranu.
- Vytvoření pomocné triangulace.
- Obnovení Delaunayovy triangulace.
- Odstranění nadbytečných trojúhelníků.




Obr. 2.14: Nalezení hran Delaunayovy triangulace protínajících povinnou hranu (převzato z [Bay]).

## 2.4 Triangulace s minimální váhou

Pro zadaný set bodů $S$ v Euklidovském prostoru je triangulace $T$. Váha triangulace $T$ je definována jakou součet všech délek hran v triangulaci $T$. Triangulace, která má minimální možnou váhu se nazývá triangulace s minimální váhou (= MWT) bodů $S$ [Mul08].



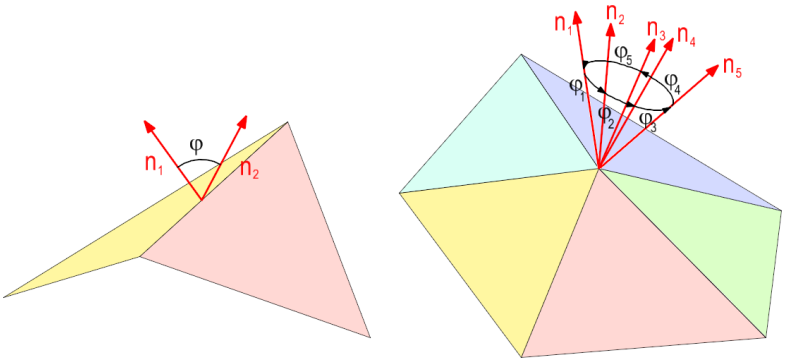

Obr. 2.15: Různé triangulace a jejich váhy (převzato z [Mul08]).

Při konstrukci je potřeba sestrojit všechny možné triangulace a vybrat z nich tu s minimální celkovou délkou hran. Existují metody pro konstrukci MWT pomocí brutální síly [Hla02]. Pro větší počet vrcholů je MWT časově velmi náročná. Metoda má exponenciální složitost. Lepší algoritmus na konstrukci není znám a není známo, zda je možné problém řešit v polynomiálním čase [Fuk06].

## 2.5 Datově závislé triangulace

Datově závislé triangulace vycházejí z některých výše uvedených triangulací (nejčastěji Delaunayovy triangulace), avšak zohledňují i jiné vlastnosti než jen vzdálenost bodů. Nejčastější takovou vlastností bývá výška nebo jiné naměřené hodnoty.

Pokud v jednotlivých bodech máme určenou výšku, obyčejná triangulace by tuto vlastnost nerespektovala. Proto je možné pro každý trojúhelník používat jako kritérium jednak vzdálenost bodů, ale navíc také odchylku normál dvou trojúhelníků přes hranu nebo odchylku normál trojúhelníků sdílející konkrétní vrchol (viz Obr. 2.16). S tímto přístupem dosahují datově závislé triangulace lepších výsledků pro terénní data než běžné triangulace [Fuk06].




Obr. 2.16: Levý obrázek ukazuje odchylku normál trojúhelníků přes hranu. Pravý obrázek ukazuje odchylku normál ve vrcholu (převzato z [Bay]).

# 3 Tetrahedronizace v $E^3$

Pro shodnou množinu bodů v prostoru se počet tetrahedronů liší v závislosti na typu tetrahedronizace. Maximální počet tetrahedronů v tetrahedronizaci lze vyjádřit pomocí následujícího vzorce [Sei95]:

$$max = O\left(N^{\lceil d/2 \rceil}\right), \tag{3.1}$$

kde $N$ je počet bodů a $d$ je rozměr dimenze (pro $E^3$ je $d = 3$). Mnohem větší problém je ovšem to, že polygon v 2D je vždy možné triangulizovat, ačkoli v 3D toto pravidlo již neplatí. Nekonvexní polyhedron nelze vždy rozložit na množinu tetrahedronů bez využití přidaných (Steinerových) bodů [Car12]. Tento rozdíl mezi 2D a 3D je důležitý pro tvorbu algoritmů v 3D. Nelze totiž obecně převést libovolný algoritmus triangulace do vyšší dimenze pro tvorbu tetrahedronizace [Sch04].

## 3.1 Delaunayova tetrahedronizace

Delaunayova triangulace se obecně snaží vyhýbat vytváření úzkých trojúhelníků a tak je výsledná trojúhelníková síť kvalitní. V dimenzi vyšší než dva je situace mnohem komplikovanější. Delaunayova tetrahedronizace [Liu05] se snaží vytvářet tetrahedrony s nejmenším poloměrem opsané koule. Tím je ovšem umožněn vznik plochých tetrahedronů, jejichž poloměr opsané koule je sice minimalizován, ovšem poloměr vepsané koule je téměř nulový.

Každý tetrahedron má jednoznačně definovanou opsanou kouli, pokud jeho vrcholy neleží v rovině. Delaunayova triangulace je taková triangulace, kde všechny tetrahedrony splňují kritérium prázdné opsané koule. Tedy žádný vrchol tetrahedronizace neleží uvnitř opsané koule tetrahedronu jehož není součástí. Tímto je Delaunayova tetrahedronizace jednoznačně definována.

Metoda na zjištění, zda vrchol $V$ leží vně, nebo uvnitř opsané koule tetrahedronu $(A, B, C, D)$ je založena na nalezení středu a poloměru opsané kružnice. Výsledný předpis pro opsanou kouli je:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2 = 0, \tag{3.2}$$

kde $[x_0 \quad y_0 \quad z_0]^T$ je střed a $r$ poloměr opsané koule. Po dosazení bodu do předpisu (3.2) lze určit, zda bod leží uvnitř podle výsledného znaménka. Pokud je rovnice (3.2) splněna, znamená to, že bod leží na povrchu koule. Pokud je výsledek levé části rovnice (3.2) záporný, leží bod uvnitř koule, pokud větší než nula, leží bod mimo kouli.

Pozici budu vůči opsané kouli tetrahedronu lze určit pomocí znaménkového testu determinantu. Předpis determinantu se získá pomocí vzorce (3.2) a jeho tvar je následující:

$$\theta = \begin{vmatrix} A_x & A_y & A_z & A_x^2 + A_y^2 + A_z^2 & 1 \\ B_x & B_y & B_z & B_x^2 + B_y^2 + B_z^2 & 1 \\ C_x & C_y & C_z & C_x^2 + C_y^2 + C_z^2 & 1 \\ D_x & D_y & D_z & D_x^2 + D_y^2 + D_z^2 & 1 \\ V_x & V_y & V_z & V_x^2 + V_y^2 + V_z^2 & 1 \end{vmatrix}, \tag{3.3}$$

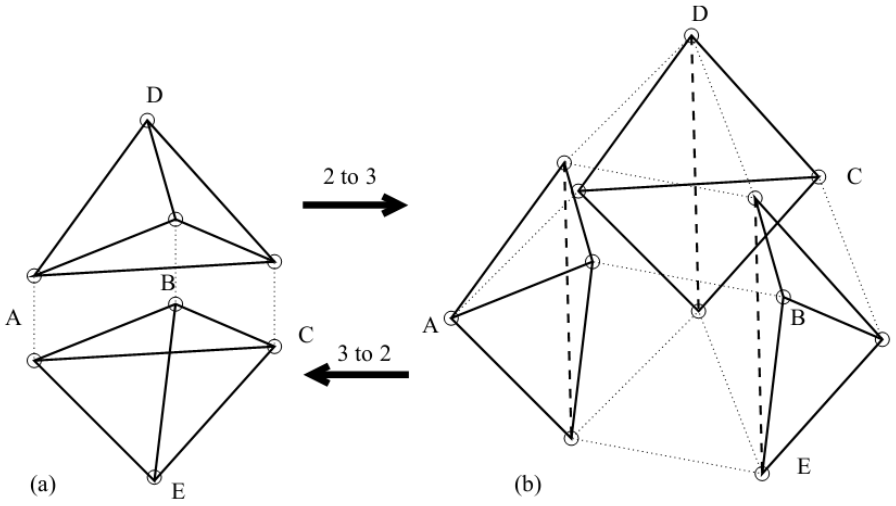
kde bod $A$ má souřadnice $[A_x \quad A_y \quad A_z]^T$, bod $B = [B_x \quad B_y \quad B_z]^T$, $C = [C_x \quad C_y \quad C_z]^T$, $D = [D_x \quad D_y \quad D_z]^T$ a testovaný bod $V = [V_x \quad V_y \quad V_z]^T$. Pokud je $\theta$ ronvo nule, leží bod $V$ na kouli, pokud je $\theta$ záporné, leží bod mimo kouli, jinak leží bod uvnitř koule.

Výpočet determinantu o velikosti $5 \times 5$ ve vzorci (3.3) lze převést na výpočet determinantu o velikosti $4 \times 4$ následujícího předpisu:

$$\theta = \begin{vmatrix} A_x - V_x & A_y - V_y & A_z - V_z & \left(A_x^2 + A_y^2 + A_z^2\right) - \left(V_x^2 + V_y^2 + V_z^2\right) \\ B_x - V_x & B_y - V_y & B_z - V_z & \left(B_x^2 + B_y^2 + B_z^2\right) - \left(V_x^2 + V_y^2 + V_z^2\right) \\ C_x - V_x & C_y - V_y & C_z - V_z & \left(C_x^2 + C_y^2 + C_z^2\right) - \left(V_x^2 + V_y^2 + V_z^2\right) \\ D_x - V_x & D_y - V_y & D_z - V_z & \left(D_x^2 + D_y^2 + D_z^2\right) - \left(V_x^2 + V_y^2 + V_z^2\right) \end{vmatrix}. \tag{3.4}$$

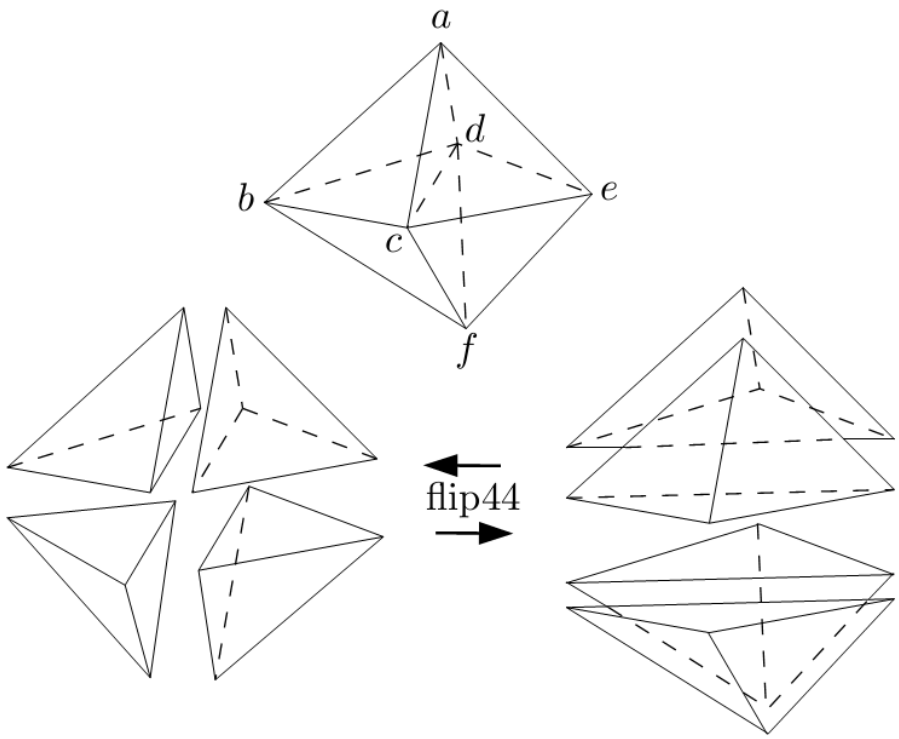
Výsledky znaménkového testu (3.4) odpovídají (3.3).

Pokud nějaký bod leží uvnitř opsané koule tetrahedronu je nutné provést transformaci (flip) tetrahedronů. Nejčastější operace, které se provádějí, jsou znázorněny na (Obr. 3.1). Transformace $\text{flip}_{32}$ převede tři tetrahedrony na dva tetrahedrony a transformace $\text{flip}_{23}$ naopak. Při transformaci $\text{flip}_{23}$ je nutné dát pozor, zda tetrahedrony společně tvoří konvexní těleso. V případě, že tomu tak není, se nesmí $\text{flip}_{23}$ provést, neboť by byl vytvořen tetrahedron, který protíná jiné tetrahedrony.




Obr. 3.1: $Flip_{32}$ a $flip_{23}$ (převzato z [Sch04]).

V případě, že dvě (nebo čtyři) stěny u dvou (nebo čtyř) tetrahedronů leží ve stejné rovině, provádí se $flip_{22}$ (nebo $flip_{44}$) (viz Obr. 3.2).

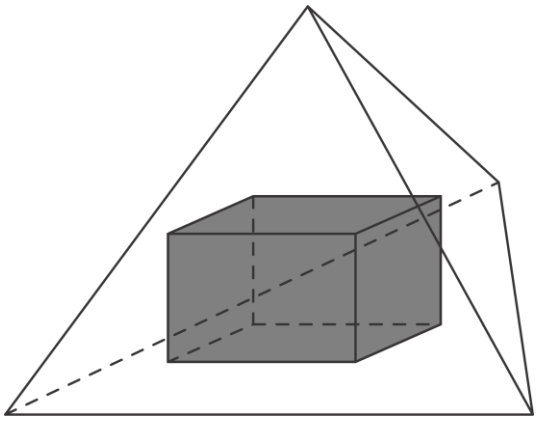



Obr. 3.2: $Flip_{44}$ (nebo $flip_{22}$ – pouze dva tetrahedrony) (převzato z [Led05]).

### 3.1.1 Inkrementální vkládání

Vložení jednoho nového bodu do Delaunayovy triangulace může změnit celou triangulaci. Toto je naštěstí pravda pouze pro některé extrémní konfigurace vrcholů. V běžném případě vložení nového bodu ovlivní pouze lokální část tetrahedronizace.

Algoritmy inkrementálního vkládání vyhledávají v každém kroku tetrahedron, do kterého lze nový bod vložit. Z tohoto důvodu je nutné mít v počátku jeden velký tetrahedron, pro který platí, že všechny později přidávané body budou uvnitř tohoto počátečního tetrahedronu (viz Obr. 3.3). Počáteční tetrahedron by se neměl dotýkat obalového kvádru bodů, tudíž je vždy vhodné vytvořit tetrahedron o několik procent větší, než tetrahedron, který se dotýká obalového kvádru bodů.



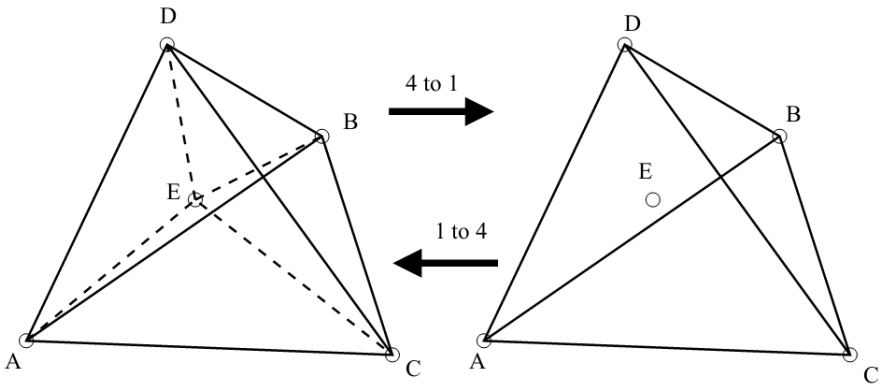
Obr. 3.3: Počáteční tetrahedron (šedě je znázorněn obalový kvádr všech bodů).

Řekněme, že máme validní Delaunayovu tetrahedronizaci o $N$ bodech. Dále si představme, že nově vkládaný bod leží uvnitř konvexní obálky $N$ bodů. Poté je možné vložení bodu do tetrohedronizace provést pomocí následujících kroků:

- Identifikace nevalidních tetrahedronů v tetrahedronizaci (jedná se o tetrahedrony, pro které platí, že přidávaný vrchol leží v jejich opsané kouli).
- Shromáždění zadních trojúhelníků nevalidních tetrahedronů (trojúhelníky, které jsou společné s validními tetrahedrony).
- Nahrazení všech nevalidních tetrahedronů pomocí nových tetrahedronů, které vzniknou spojením nashromážděných trojúhelníků s přidávaným vrcholem.

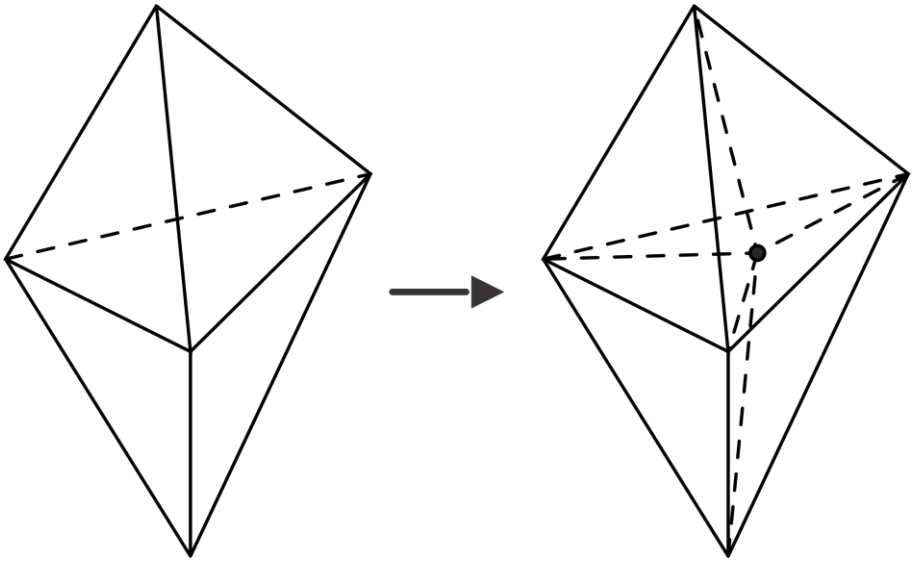
Tento inkrementální algoritmus se nazývá Bowyer-Watsonův algoritmus. Poté co byly nalezeny všechny nevalidní tetrahedrony je výpočetní náročnost již velmi nízká a je lineárně úměrná počtu nevalidních tetrahedronů. V praxi se nejprve nalezne tetrahedron, který obsahuje přidávaný bod ve své opsané kouli. Ostatní nevalidní tetrahedrony jsou již hledány pomocí prohledávání sousedů a zjišťování jejich validity. Výsledek tohoto postupu je Delaunayova triangulace o $(N + 1)$ bodech.

Dalším algoritmem inkrementálního vkládání je Green-Sibsonův algoritmus potřebující také počáteční tetrahedron (viz Obr. 3.3). V každém kroku je nalezen tetrahedron, který obsahuje přidávaný bod. Pokud je přidávaný bod uvnitř nalezeného tetrahedronu je provedeno jeho vložení pomocí operace $\text{flip}_{14}$ (viz Obr. 3.4).



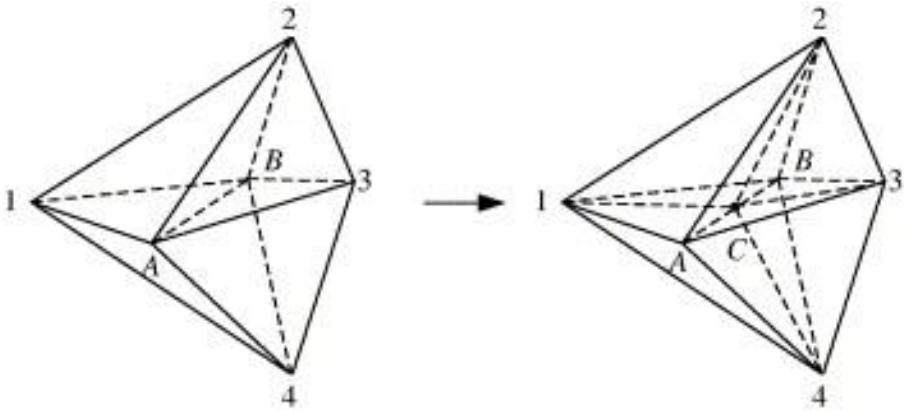

Obr. 3.4: Vložení bodu uvnitř tetrahedronu (převzato z [Sch04]).

V případě, že přidávaný bod neleží přímo uvnitř tetrahedronu, je nutné zjistit, zda leží na hraně nebo ve stěně tetrahedronu. Pokud se nachází bod ve stěně tetrahedronu je nutné jeho vložení provést pomocí operace $flip_{26}$ (viz Obr. 3.5). Tímto vložením se každý ze dvou sousedních tetrahedronů rozdělí na tři nové tetrahedrony. Výsledně tedy vznikne nových šest tetrahedronů ze dvou původních.



Obr. 3.5: Vložení bodu ve stěně.

Pokud se vkládaný bod nachází na hraně tetrahedronu, nebo přesněji tetrahedronů, je nutné jeho vložení provést pomocí operace $flip_{k\text{-}2k}$ (viz Obr. 3.6). Touto operací vložení se z $k$ tetrahedronů obsahujících společnou hranu, na které leží přidávaný bod, stane $2k$ nových tetrahedronů.




Obr. 3.6: Vložení bodu na hraně (převzato z [Dai05]).

Poslední možná situace, která může nastat při vkládání bodu to tetrahedronizace ja taková, že přidávaný bod v tetrahedronizaci již existuje. V takovém případě se tento bod do tetrahedronizace již nepřidává, neboť je v ní již obsažen.

Po vložení bodu do tetrahedronizace není zaručeno, že výsledná tetrahedronizace je opět Delaunayova. Z tohoto důvodu je nutné provést legalizaci nově vytvořených tetrahedronů a jejich sousedů pomocí operací $flip_{23}$ a $flip_{32}$ (viz Obr. 3.1). Aby byla tetrahedronizace opět Delaunayovská, musí být pro všechny tetrahedrony splněno Delaunayovo kritérium. To znamená, že žádný vrchol, který není součástí tetrahedronu nesmí být obsažen v opsané kouli tetrahedronu.

Pokud již byly do tetrahedronizace vloženy všechny vstupní body, je nyní nutné odebrat čtyři přidané body, které definují velký obalový tetrahedron. Postupně je tedy nutné projít všechny tetrahedrony a zjistit, zda je alespoň jeden z vrcholů vrcholem počátečního tetrahedronu. Pokud je tomu tak, je tento tetrahedron odebrán z tetrahedronizace. Po odebrání všech těchto tetrahedronů je získána výsledná Delaunayova tetrahedronizace.

Algoritmickou složitost Delaunayovy tetrahedronizace pro nejhorším případ je možné vyjádřit pomocí vzorce:

$$max = O\left(N^{\left\lfloor {(d+1)}/{2} \right\rfloor + 1}\right), \tag{3.5}$$

kde pro $E^3$ je $d = 3$ a tudíž algoritmická složitost pro nejhorší případ je $O(N^3)$ [KoJ05]. Pomocí vhodné implementace a při vhodném rozložení vstupních bodů lze získat očekávanou časovou složitost $O(N \log N)$.

### 3.1.2 Paralelní metody

Obdobně jako u triangulace existují i pro tetrahedronizaci různé paralelní metody. V následujících kapitolách budou některé tyto metody popsány.

#### 3.1.2.1 De-Wall

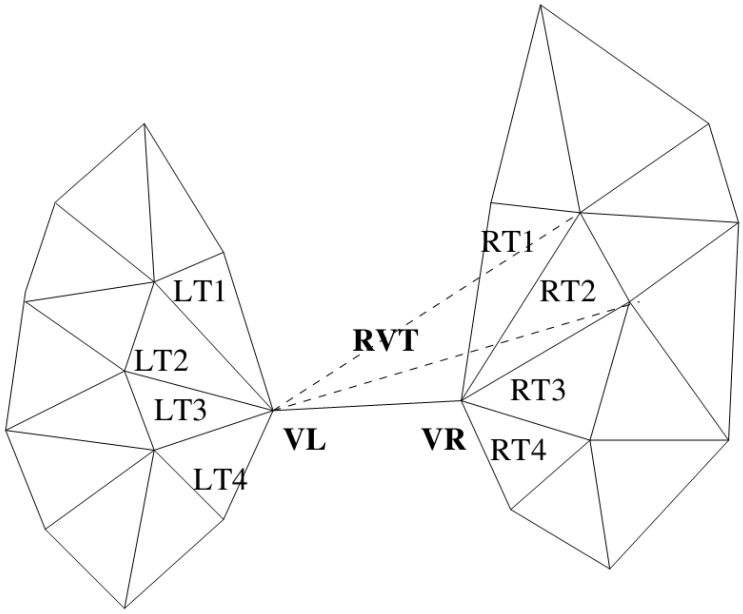
Tetrahedronizace pomocí algoritmu De-Wall je téměř shodná postupem v $E^2$. Jedná se tedy o pouhé rozšíření algoritmu do $E^3$. Místo dělící přímky je využívána dělící rovina $\alpha$, která je cyklicky měněna tak, že je rovnoběžná s rovinou $\rho_{yz}, \rho_{xz}, \rho_{xy}$.

#### 3.1.2.2 Min-Bin Chenova metoda

Algoritmus paralelní tetrahedronizace popsal Min-Bin Chen v článku [Che11]. První fází algoritmu je rozdělení vstupních bodů. Používaný algoritmus se snaží co nejvíce zmenšit povrchy hranic, které bude zapotřebí spojit. Postup je takový, že jsou body nejprve rozděleny podle $x$-ové souřadnice do skupin. V dalším kroku je každá množina bodů rozdělena podle $y$-ové souřadnice a v posledním kroku podle $z$-ové souřadnice. Při každém dělení skupin se využívá mediánu, takže výsledné skupiny jsou stejně rozsáhlé.

Každá množina bodů je nezávisle tetrahedronizována pomocí algoritmu inkrementálního vkládání. Výpočet je tedy prováděn paralelně. Po dokončení dílčích tetrahedronizací je nutné všechny tetrahedronizace spojit dohromady. Pro spojení tetrahedronizací jsou nalezeny takzvané „ovlivněné zóny“. Jedná se o tetrahedrony, které mohou být při spojování pozměněny pomocí operací $\text{flip}_{ij}$. Hledání těchto tetrahedronů se provádí postupně směrem od konvexní obálky směrem dovnitř tetrahedronizace.

Při spojování jednotlivých zón je nejprve nutné vytvořit první tetrahedron, který vznikne pomocí jednoho bodu z jedné tetrahedronizace a trojúhelníku z konvexní obálky druhé tetrahedronizace (viz Obr. 3.7). V dalších krocích se vytvářejí další tetrahedrony tak, že se vždy využije jeden trojúhelník z jedné tetrahedronizace, bod z druhé tetrahedronizace a minimálně jeden trojúhelník z již vytvořených spojových tetrahedronů. Při vytvoření každého tetrahedronu je nutné zkontrolovat, zda je dodrženo Delaunayovo kritérium a případně tetrahedronizaci transformovat.




Obr. 3.7: První tetrahedron mezi dvěma tetrahedronizacemi (převzato z [Che11]).

Spojování jednotlivých tetrahedronizací se provádí v opačném pořadí, než ve kterém byla vstupní množina bodů postupně dělena na podmnožiny bodů. Jedná se tedy o typický paralelní přístup „rozděl & panuj“. Algoritmická složitost tohoto paralelního algoritmu je v očekávaném případě $O(N \log N)$.

#### 3.1.2.3 Modifikace inkrementálního vkládání

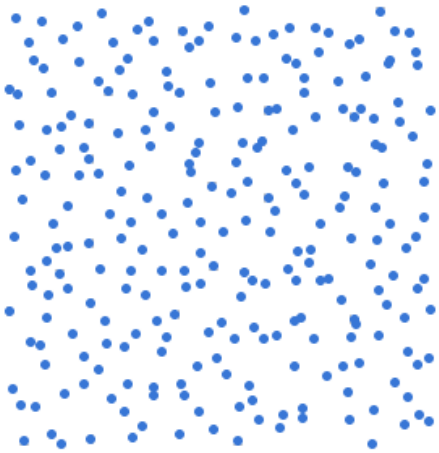
Tato metoda paralelní tetrahedronizace je rozšířením paralelní triangulace popsané v (kap. 2.1.4.2) z $E^2$ do $E^3$ [Koh02] [Koh05].

# 4 Paralelní triangulace v $E^2$

Hlavním cílem nově vytvářené metody triangulace je co nejnižší časová náročnost. Myšlenka urychlení je taková, že pokud se vstupní data rozdělí do několika oddělených množin a ty se ztriangulizují zvlášť, musí být celková časová náročnost nižší. Navíc lze využít paralelního přístupu. Při triangulaci se využívá principu rozděl a panuj. V následujících kapitolách bude popsán postup nově navrhnuté metody paralelní triangulace [Ska12].

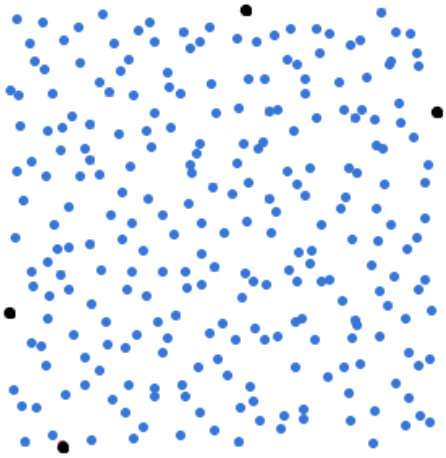
## 4.1 Rozdělení bodů

Vstupem pro metodu paralelní triangulace je množina náhodně rozložených bodů v rovině o souřadnicích $[x, y]^T$ (viz Obr. 4.1).



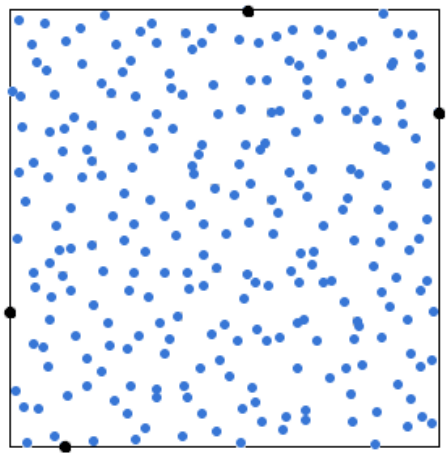
Obr. 4.1: Množina vstupních bodů v rovině.

Aby bylo možné dělit množinu bodů na několik podskupin je nutné nejprve zjistit její Bounding Box (= obalový obdélník). Pro nalezení tohoto obdélníku je nutné zjistit body s maximální a minimální $x$-ovou a $y$-ovou složkou (viz Obr. 4.2). Složitost nalezení těchto bodů je $O(N)$.



Obr. 4.2: Body s maximální a minimální x-ovou a y-ovou složkou.

Při nalezení čtyř hraničních bodů je již možné si kolem vstupní množiny bodů v rovině představit obalový obdélník (viz Obr. 4.3).

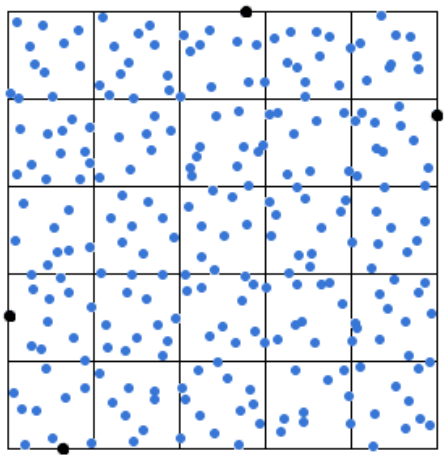


Obr. 4.3: Bounding Box.

Pro rozdělení bodů do pravidelné mřížky je nutné určit velikost hrany $dx$ a $dy$. Velikost těchto hran je nutné určit s ohledem na předpokládaný počet bodů v každé buňce nebo na požadovaný počet buněk. Rozdělení bodů do příslušných buněk, které jsou očíslovány souřadnicemi $\{i, j\}$ podobně jako 2D matice, se provádí podle hashovacího vzorce

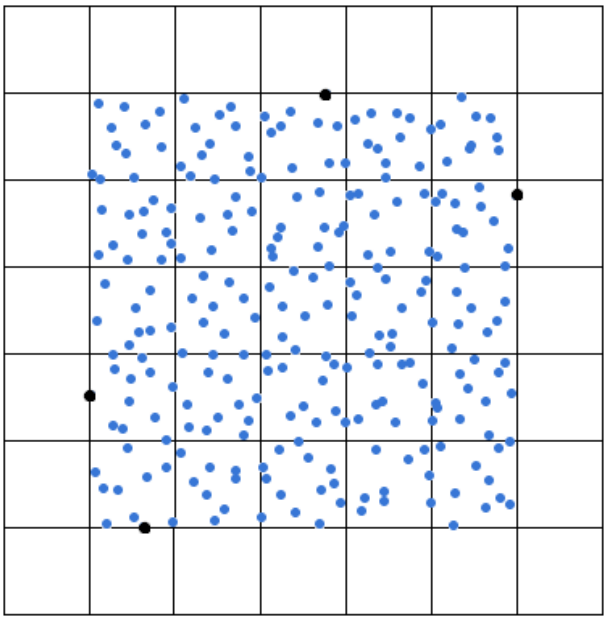
$$\begin{aligned} i &= \left\lfloor \frac{(x - x_{min})}{dx} \right\rfloor \\ j &= \left\lfloor \frac{(y - y_{min})}{dy} \right\rfloor \end{aligned} \tag{4.1}$$

kde $x$, $y$ jsou souřadnice bodu, $x_{min}$ je x-ová souřadnice levé strany AABB, $y_{min}$ je y-ová souřadnice spodní strany AABB. Výsledné rozdělení bodů do jednotlivých buněk je zobrazeno na (Obr. 4.4).



Obr. 4.4: Rozdělení bodů do pravidelné mřížky.

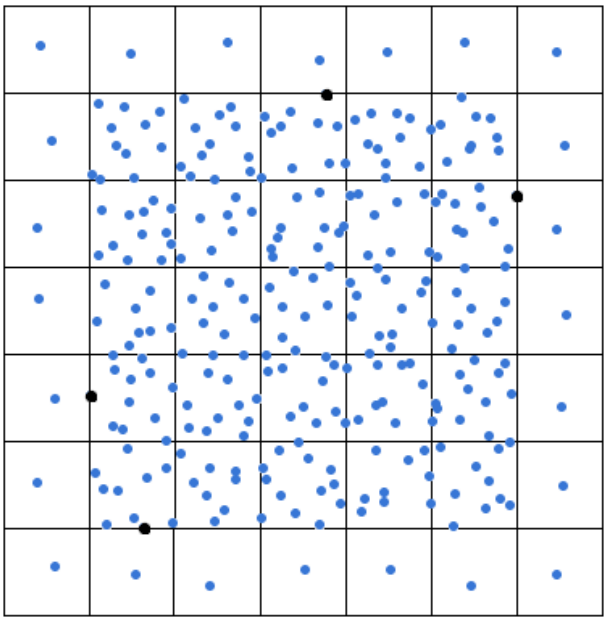
Z důvodu vytvoření konvexní obálky triangulace se body rozdělené do pravidelné mřížky ještě obklopí jednou vrstvou prázdných gridů. Po spojení triangulací bude totiž zaručeno, že konvexní obálka vstupní množiny bodů bude uvnitř spojených triangulací a její následné získání bude jednodušší. Výsledné zobrazení gridů s body je na (Obr. 4.5).



Obr. 4.5: Přidané gridy okolo vytvořené pravidelné mřížky.

V dalším kroku je nutné zkontrolovat, zda každý grid obsahuje alespoň jeden vnitřní bod, který by se měl triangulizovat. Pokud tomu tak není, je do střední části

gridu vložen náhodný bod. Rozmístění přidaných bodů je zobrazeno na (Obr. 4.6). Tím že se nevkládají body přímo do středu gridu, ale částečně náhodně, zvyšuje numerickou stabilitu při spojování jednotlivých triangulací.

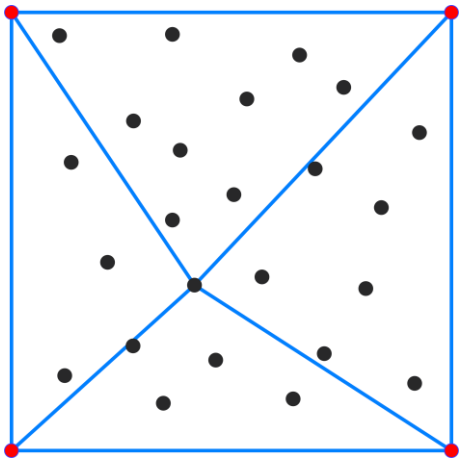


Obr. 4.6: Vložené náhodné body do prázdných gridů.

## 4.2 Triangulace jedné množiny bodů

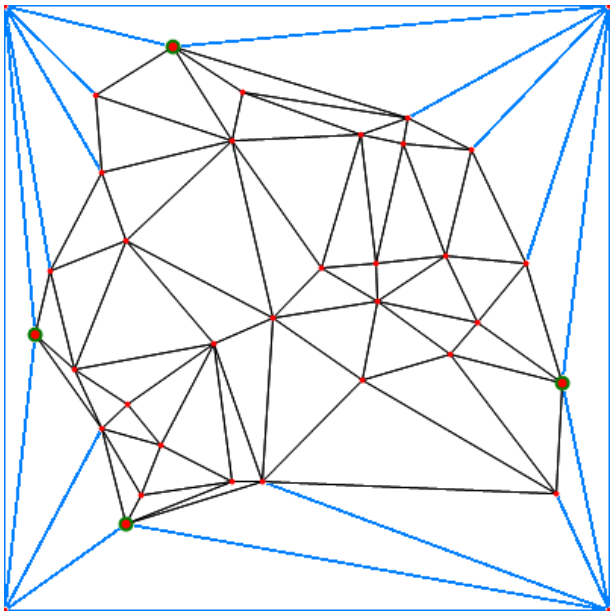
Vstup pro triangulaci jedné buňky jsou body v ní obsažené a navíc čtyři body definující hraniční obdélník (přidané body). Běžná Delaunayova triangulace pomocí inkrementálního vkládání nejprve vytvoří velký obalový trojúhelník. Při této triangulaci ovšem tento trojúhelník není zapotřebí vytvářet, neboť je znám obalový obdélník všech vnitřních bodů.

V množině bodů se vybere jeden náhodný. Pomocí tohoto bodu a čtyř přidaných bodů do množiny bodů se vytvoří čtyři počáteční trojúhelníky (viz Obr. 4.7). V dalším kroku se pokračuje s běžnou Delaunayovo triangulací pomocí inkrementálního vkládání všech zbylých bodů do předpřipravené počáteční triangulace.



Obr. 4.7: Počáteční triangulace jedné buňky.

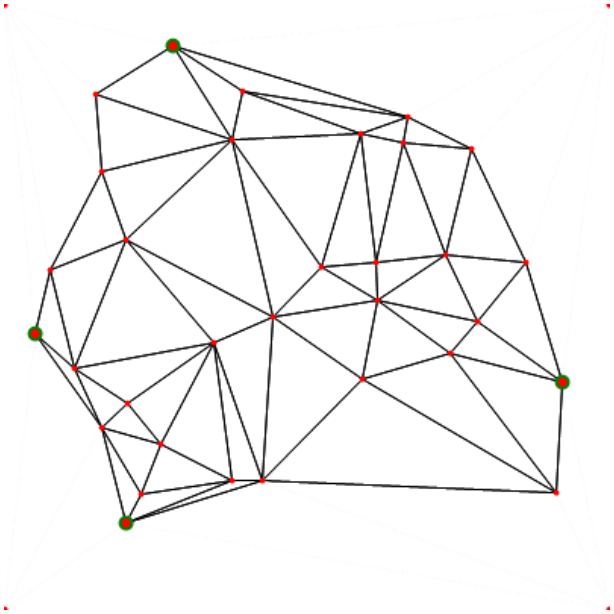
Po dokončení Delaunayova inkrementálního vkládání bodů do triangulace se získá triangulizovaný obdélník (viz Obr. 4.8). Vlastnosti, které lze využít, je, že každá hrana obalového obdélníku obsahuje právě jeden trojúhelník. Vrcholy těchto čtyř trojúhelníků, které jsou z množiny vstupních bodů, je nutné nalézt (viz tučně označené body v Obr. 4.8). Použití Delaunayovy triangulace není nutnou podmínkou. Pro triangulaci jednotlivých gridů je možné využít jakoukoliv metodu triangulace.



Obr. 4.8: Triangulizovaný obdélník.

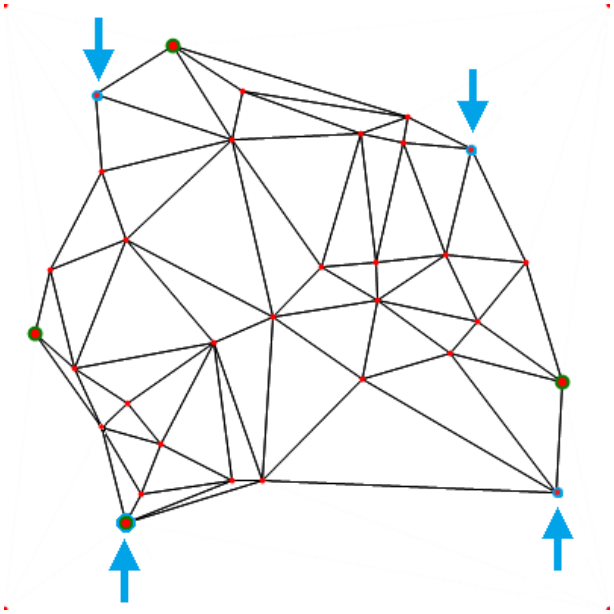
V dalším kroku jsou odebrány všechny hrany, které obsahují bod definující obalový obdélník (světle označené hrany na Obr. 4.8 a výstup viz Obr. 4.9). Při odebírání hran je potřeba vědět, k jakému bodu ze čtyř přidaných jsou dané dvě hrany trojúhelníku napojeny. Třetí hrana musí být se svou správnou orientací přidána do

seznamu hran patřící k danému přidanému rohovému bodu. Po dokončení odebírání hran se pomocí seznamu orientovaných hran vytvoří posloupnost bodů, které definují hranici triangulace.



Obr. 4.9: Triangulace s odebranými hranami obsahující přidané rohové body.

Ze získaných bodů, které tvoří hranici trojúhelníkové sítě, se dále nalezne vždy jeden nejbližší bod ke každému ze čtyř bodů definujících obalový obdélník. Tyto body jsou zobrazeny na (Obr. 4.10).



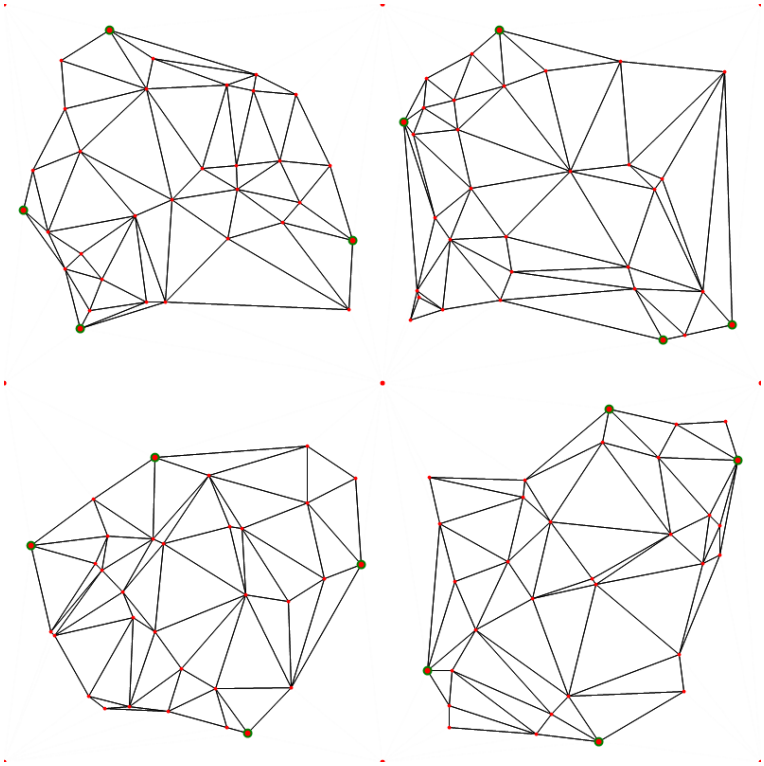
Obr. 4.10: Zvýrazněné body, které jsou nejbližší k přidaným rohovým bodům.

## 4.3 Spojení jednotlivých množin

Ve výše uvedených podkapitolách bylo vysvětleno, jak je možné vstupní množinu rozdělit do jednotlivých podmnožin, které lze poté odděleně triangulizovat. Z toho získáme triangulace jednotlivých podmnožin, které ovšem nebudou mezi sebou propojené. Dalším postupem je tedy sešití jednotlivých triangulací dohromady tak, aby utvořily výslednou triangulaci.

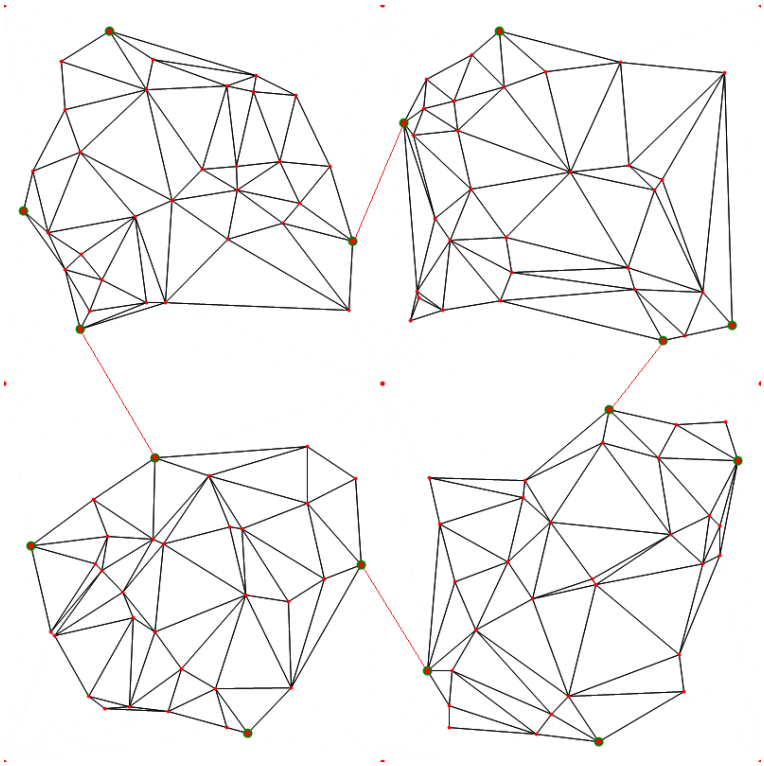
Jak víme z (kap. 4.2), z každé triangulizované podmnožiny získáme čtyři body, které tvořily krajní trojúhelníky. Dále získáme čtyři body, které byly nejblíže k pomocným rohovým bodům. Všechny tyto body mají své charakteristické vlastnosti, které budou dále využity k sestavení výsledné triangulace.

Jednotlivé triangulizované podmnožiny se budou sešívat po čtyřech, neboť tím utvoří v obecném případě nekonvexní polygon, který je možné opět odděleně triangulizovat, a tím využít paralelizace i při sešívání. Vstupem pro sešití vnitřku triangulace je tedy čtveřice triangulizovaných podmnožin, které se budeme snažit sešít (viz Obr. 4.11).



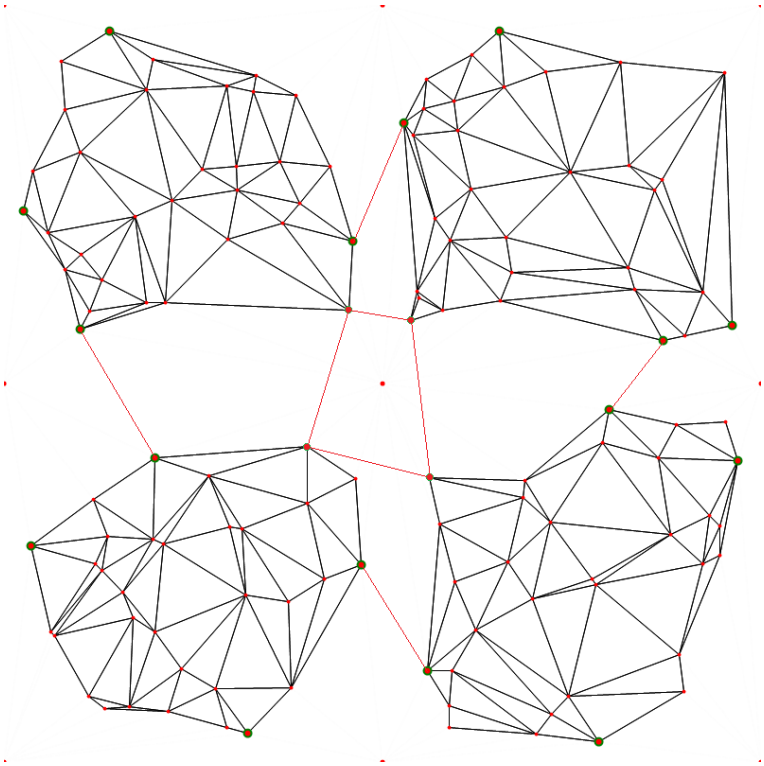
Obr. 4.11: Příklad jednotlivých triangulizovaných podmnožin.

Při sešívání vnitřku triangulace nejprve využijeme body, které tvořily krajní trojúhelník. Tyto body je totiž možné se sousedními triangulizovanými podmnožinami jednoduše propojit, a tím oddělíme jednotlivé vnitřní díry, které budeme dále vyplňovat. Příklad propojení bodů je vidět na (Obr. 4.12).



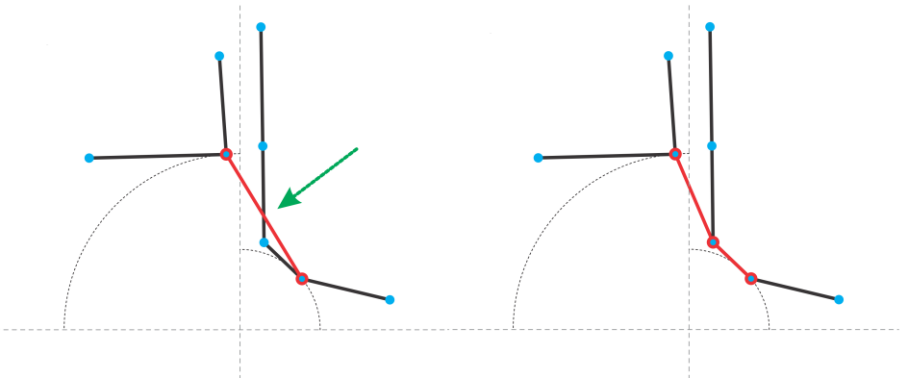
Obr. 4.12: Propojení bodů, které tvořily krajní trojúhelník.

Tímto způsobem nám vznikla hranice nekonvexního polygonu, který potřebujeme triangulizovat. Užitečnou vlastností vzniklého polygonu je, že se jedná o polygon hvězdicovitého tvaru se čtyřmi rameny. Této vlastnosti je vhodné využít pro retriangulaci polygonu. Nyní využijeme nejbližší body, které lze v nejjednodušším případě propojit čtyřmi hranami (viz Obr. 4.13).



Obr. 4.13: Propojení nejbližších bodů hranami.

Zda lze nejbližší body propojit pouze čtyřmi hranami je nutné nejprve otestovat. Test probíhá tak, že si danou hranu mezi nejbližšími body pomyslně vytvoříme a testujeme průsečík této hrany s ostatními hranami, které mohou kolizi vyvolat (vždy to jsou ty řetězce hran, které daný nejbližší vrchol obsahují). V případě, že průsečík s hranami hranice neexistuje, je možné danou úsečku vytvořit. Pokud ovšem průsečík existuje, je nutné danou hranu přidat do seznamu definující vnitřní polygon (viz Obr. 4.14). Stejný princip testování uděláme pro všechny čtyři hrany.

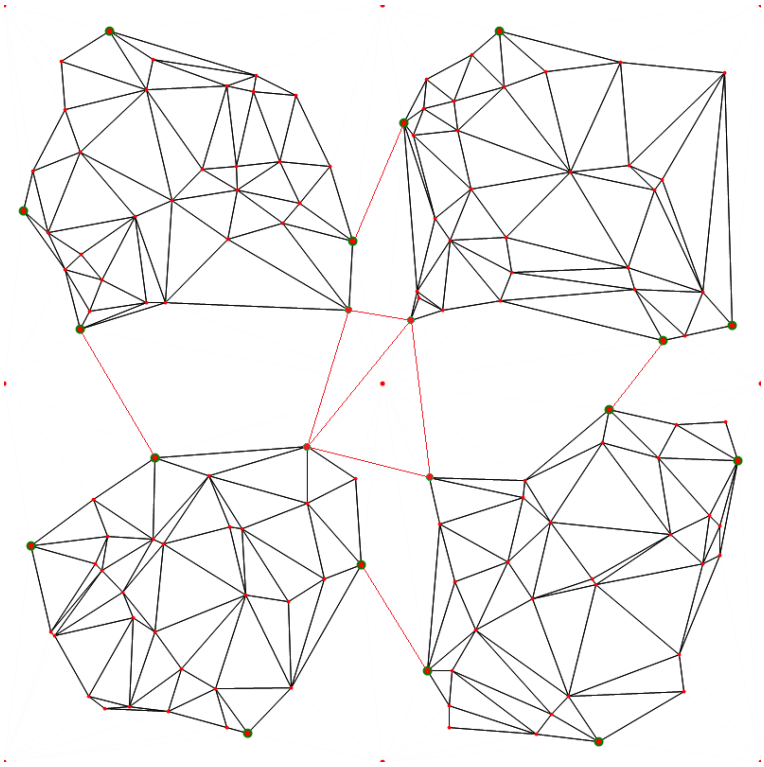


Obr. 4.14: Levý obrázek ukazuje průsečík pomyslné červené hrany s řetězcem hran. Pravý obrázek ukazuje, jak hranu zařadit do seznamu definující vnitřní polygon.

Tímto způsobem jsme původní nekonvexní polygon opět rozdělili na pět oddělených částí, které můžeme triangulizovat samostatně (rozdělení viz Obr. 4.13).

### 4.3.1 Triangulace středu hvězdice

V nejjednodušším případě má vnitřní polygon pouze čtyři hrany (tzn., že neexistují žádné průsečíky). Tento polygon má pouze dvě možnosti, jak jej triangulizovat. Přidáme tedy takovou hranu ze dvou možných, která má menší délku. Tím získáme rovnostrannější trojúhelníky (viz Obr. 4.15).



Obr. 4.15: Triangulace vnitřní části nekonvexního polygonu.

V případě, že vnitřní polygon obsahuje více hran než jen čtyři je nutné díru triangulizovat jiným způsobem. Algoritmus, který je využit pro triangulaci je nazýván odřezávání uší. Hranice díry je postupně procházena dokola a je testováno, zda je možné z dvou po sobě následujících hran vytvořit trojúhelník. K tomuto testování je využit znaménkový test orientace:

$$\theta = \begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix}, \tag{4.2}$$

kde body $A = [A_x, A_y]^T$, $B = [B_x, B_y]^T$ a $C = [C_x, C_y]^T$ jsou tři po sobě jdoucí body hranice. Pokud jsou vytvářeny trojúhelníky s orientací ve směru hodinových ručiček, pak je možné trojúhelník vytvořit pokud je výsledné $\theta$ ze vzorce (4.2) záporné. Při sestavování trojúhelníků s orientací proti směru hodinových ručiček musí být $\theta$ větší než nula.

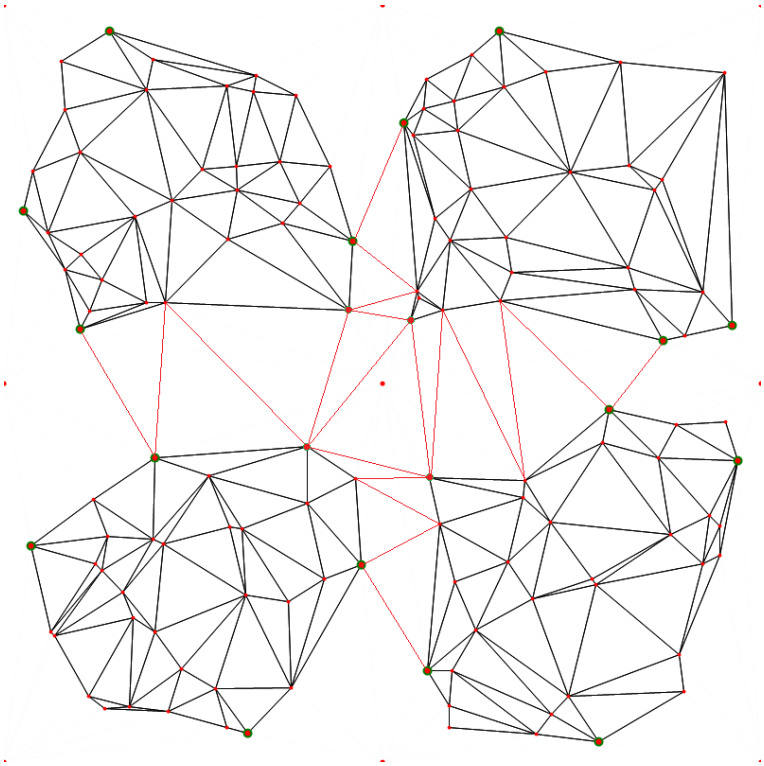
Uši, nebo-li trojúhelníky, jsou odřezávány do doby, něž zůstanou pouze čtyři body hranice. V tomto okamžiku se využije postup jako při triangulaci díry pouze o čtyřech vrcholech (postup popsaný výše).

### 4.3.2 Triangulace ramen hvězdice

Další postup algoritmu je triangulace zbývajících částí původního nekonvexního polygonu, tak zvaných ramen hvězdice. Zde si lze všimnou toho, že jednotlivé části jsou vždy monotónní buď v jedné či druhé ose. Toho lze s výhodou využít a je možné aplikovat algoritmus triangulace monotónního polygonu, který je uveden níže (převzato z [Kol]):

*{ **C** – řetězec vrcholů, vršek **t**, dno **b**, přidává se na dno,*
***v** – aktuální vrchol, **v+** sousední vrchol a je nad **v** }*
*Seřadit vrcholy sestupně/vzestupně podle monotónní osy*
***C** ← 2 nejvyšší vrcholy, **v** ← 3. nejvyšší vrchol*
***while v !=** nejnižší vrchol do*
    ***case 1:** // **v** je v 2. řetězci než je **C***
        *vést diagonálu z **v** do **t+1***
        *odstranit z **C** **v+***
        ***if** |**C**| < **2 then** přidat **v** do **C**, **v** ← next(**v**)*
    ***case 2:** // **v** sousedí se dnem **C***
        ***case 2.a** // **v+** je ostře konvexní*
            *vést diagonálu z **v** do **b-1***
            *odstranit z **C** **v+***
            ***if** |**C**| < **2 then** přidat **v** do **C**, **v** ← next(**v**)*
        ***case 2.b** // **v+** není konvexní*
            *přidat **v** do **C**, **v** ← next(**v**)*

Nyní máme triangulizovaný celý původní nekonvexní polygon. Výsledek je možné vidět na (Obr. 4.16). Jak již bylo řečeno, tento postup je možné aplikovat odděleně pro všechny díry a výpočet provádět paralelně.



Obr. 4.16: Hotová triangulace nekonvexního polygonu.

## 4.4 Odebrání přidaných bodů

Při rozdělování vstupních bodů do jednotlivých gridů mřížky je možné, že některé gridy zůstanou prázdné. V takových případech bylo do těchto gridů přidáno vždy po jednom vygenerovaném novém bodu. V tomto okamžiku jsou již všechny částečné triangulace spojeny a je nutné odebrat všechny přidané body, které nebyly součástí vstupní množiny bodů pro triangulaci.

Odebíraný bod může ležet buďto uvnitř triangulace a poté je po jeho odebrání nutné vyplnit vzniklou díru. Druhý případ je, když odebíraný bod leží na okraji triangulace. V takovém případě je nutné triangulizovat hranici triangulace a vytvořit konvexní obal hranice.

Zjištění, zda se odebíraný bod nachází uvnitř nebo na okraji triangulace, se provádí pomocí testování hranice díry, která vznikne po odebrání všech trojúhelníků obsahujících odebíraný bod. Pokud je seřazená orientované hranice uzavřená, nebo-li první a poslední bod hranice je shodný, pak se jednalo o bod, který byl uvnitř triangulace. V opačném případě se jednalo o bod na okraji triangulace.

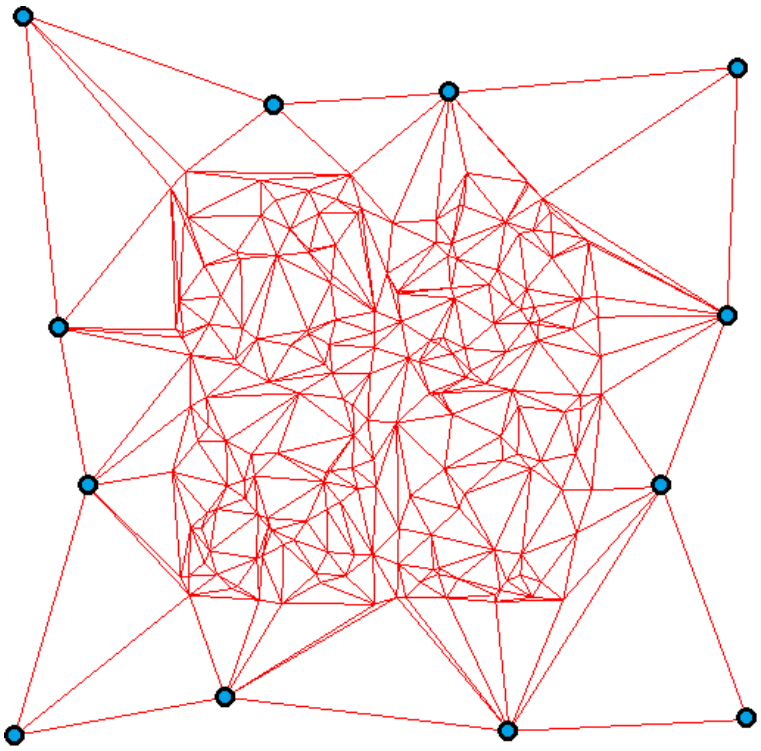
### 4.4.1 Odebrání bodu uvnitř triangulace

Postup pro odebrání bodu z triangulace je následující. Nejprve jsou z triangulace odebrány všechny trojúhelníky, které obsahují odebíraný bod, a zároveň se ukládá hranice vytvořené díry. Po odebrání všech trojúhelníků obsahujících daný bod je získána seřazená orientovaná hranice vzniklé díry v triangulaci.

Algoritmus využívaný pro triangulaci díry je stejný jako algoritmus pro triangulaci středu hvězdice (viz kap. 4.3.1).

### 4.4.2 Odebrání bodu na okraji triangulace

Před samotným odebíráním přidaných bodů na okraji triangulace má dočasná triangulace podobu zobrazenou na (Obr. 4.17). Je názorně vidět, že všechny přidané body na okraji jsou za budoucí konvexní obálkou výsledné triangulace. Postupným odstraňováním těchto bodů se tedy bude postupně vytvářet konvexní obálka finální triangulace.



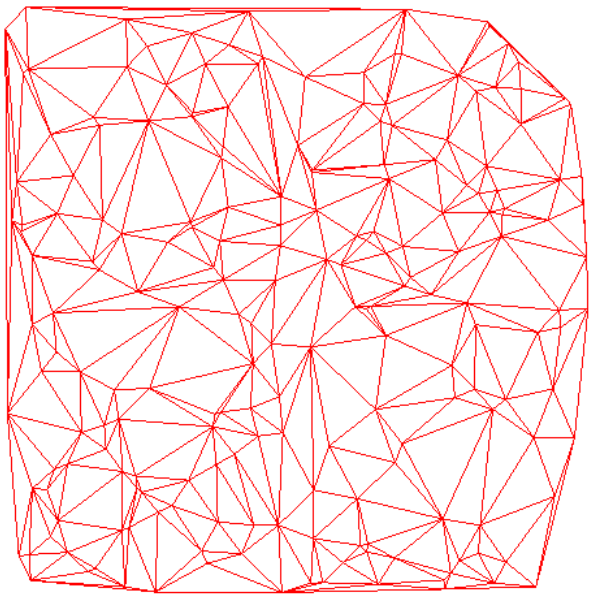
Obr. 4.17: Triangulace s přidanými body na okrajích (zvýrazněny tučně).

Při odebírání bodu na okraji triangulace jsou nejprve odstraněny všechny trojúhelníky obsahující odebíraný bod a zároveň je sestavována hranice vzniklá odebráním trojúhelníků. Po odstranění všech potřebných trojúhelníků je získána orientovaná hranice, kterou je nyní nutné triangulizovat.

Algoritmus pro triangulaci hranice je podobný algoritmu pro triangulaci díry (viz kap. 4.3.1). Jedná se opět o algoritmus ořezávání uší. Postupně se prochází celá hranice od počátku a testuje se, zda lze odříznout ucho a vytvořit tak nový trojúhelník. K testování se využívá znaménkového testu pro správnou orientaci trojúhelníku (4.2). Při přidání trojúhelníku je nutné aktualizovat hranici. Z původní hranice je odebrán jeden bod, který je společný pro dvě hrany hranice, které byly požity pro vznik nového trojúhelníku. Poté co se dojde na konec hranice, je aktuální hranice testována znovu od

počátku. Algoritmus končí v případě, že během jednoho celého průchodu nebyl vytvořen ani jeden nový trojúhelník nebo pokud hranice obsahuje pouze dva body.

Postupným odebráním všech okrajových bodů se vytvoří konvexní obálka triangulace. Vzniklá triangulace i s její konvexní obálkou je zobrazena na (Obr. 4.18).



Obr. 4.18: Výsledná triangulace s vytvořenou konvexní obálkou.

# 5 Paralelní tetrahedronizace v $E^3$

V předchozí kapitole byla popsána nově navržená metoda pro paralelní triangulaci množiny bodů v $E^2$. V této kapitole bude popsána metoda pro paralelní tetrahedronizaci množiny bodů v $E^3$ [Ska12]. Hlavní princip metody je podobný jako pro navrženou triangulaci.

## 5.1 Rozdělení bodů

Vstupem pro tetraheronizaci je množina bodů v prostoru. Pro paralelní tetrahedronizaci je nutné tuto množinu bodů rozdělit do několika nezávislých gridů. Počet gridů a zároveň tedy i počet dělení v každé ose je vypočteno podle vzorce

$$počet\ dělení = \sqrt[3]{\frac{počet\ bodů}{počet\ bodů\ na\ grid}}, \tag{5.1}$$

kde $počet\ bodů\ na\ grid$ je proměná, jejíž optimální hodnota budede nalezena ve fázi testování tetrahedronizace.

V dalším kroku je nejprve nutné najít tzv. min max box, což je obalový hranol, který ve svém objemu obsahuje všechny vstupní body. Pro nalezení min max boxu je nutné nalézt minimum a maximum v každé ze tří souřadnic $x$, $y$ a $z$.

Po nalezení minimálních a maximálních hodnot ve všech souřadnicích je vhodné tyto hodnoty od sebe na všech osách mírně oddálit, neboli mírně nafouknout min max box. Zvětšení je prováděno z důvodu větší numerické stability tetrahedronizace. Rozšíření se provádí vždy o 0,1% každém směru. Je tedy nejprve nutné vypočíst po jakých rozsazích by se body rozdělily do jednotlivých gridů mřížky. Výpočet je proveden podle následujícího vzorce

$$\{dx, dy, dz\} = \left\{\frac{x_{\max} - x_{\min}}{počet\ dělení}, \frac{y_{\max} - y_{\min}}{počet\ dělení}, \frac{z_{\max} - z_{\min}}{počet\ dělení}\right\}, \tag{5.2}$$

kde $dx$, $dy$ a $dz$ je délka hrany gridu v souřadnicích $x$, $y$ a $z$ a $počet\ dělení$ je vypočten pomocí vzorce (5.1). V dalším kroku je nutné rozšířit obalový min max box, což se provádí pomocí vzorce

$$\begin{gathered}\{\varepsilon_x, \varepsilon_y, \varepsilon_z\} = \{0{,}001 \cdot dx, 0{,}001 \cdot dy, 0{,}001 \cdot dz\} \\ \begin{aligned}\{x_{\min}, x_{\max}\} &= \{x_{\min} - \varepsilon_x, x_{\max} + \varepsilon_x\} \\ \{y_{\min}, y_{\max}\} &= \{y_{\min} - \varepsilon_y, y_{\max} + \varepsilon_y\} \\ \{z_{\min}, z_{\max}\} &= \{z_{\min} - \varepsilon_z, z_{\max} + \varepsilon_z\}.\end{aligned}\end{gathered} \tag{5.3}$$

Po provedení změny minimálních a maximálních souřadnic podle vzorce (5.3) je znovu přepočteno po jakých rozsazích se budou body rozdělovat do gridů. Pro výpočet je využit vzorec (5.2).

Jeden bod po druhém je nutné projít a přiřadit do správného gridu v pravidelné mřížce. Souřadnice gridu jsou vypočteny podle vzorce

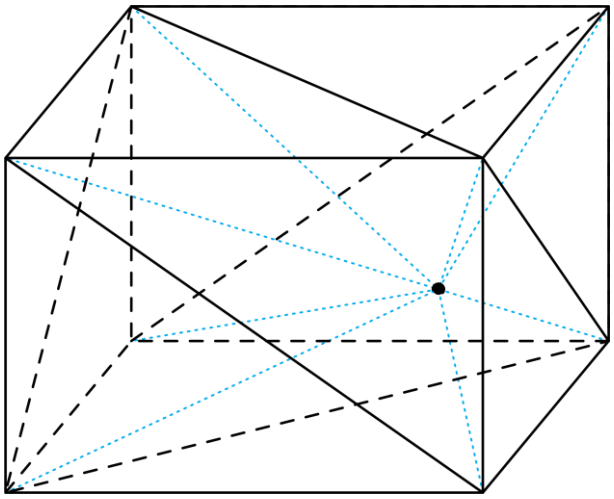
$$\begin{aligned} i &= \left\lfloor \frac{(x - x_{min})}{dx} \right\rfloor \\ j &= \left\lfloor \frac{(y - y_{min})}{dy} \right\rfloor \\ k &= \left\lfloor \frac{(z - z_{min})}{dz} \right\rfloor. \end{aligned} \tag{5.4}$$

Po roztřídění bodů do jednotlivých gridů podle (5.4) je nutné vygenerovat body, které tvoří pravidelnou mřížku gridů a jsou vždy hraničními body pro každý grid. Vygenerované body je nutné také přiřadit vždy správnému gridu, ve kterém budou později využívány k vytvoření obalového kvádru.

## 5.2 Tetrahedronizace jedné množiny bodů

Vstupem je množina bodů, které náleží do shodného gridu. Tetrahedronizace jednotlivých množin bodů je možné provádět současně, neboť jsou na sobě nezávislé. Navíc není při paralelní tetrahedronizaci gridů nutná žádná vnitřní komunikace mezi jednotlivými tetrahedronizacemi.

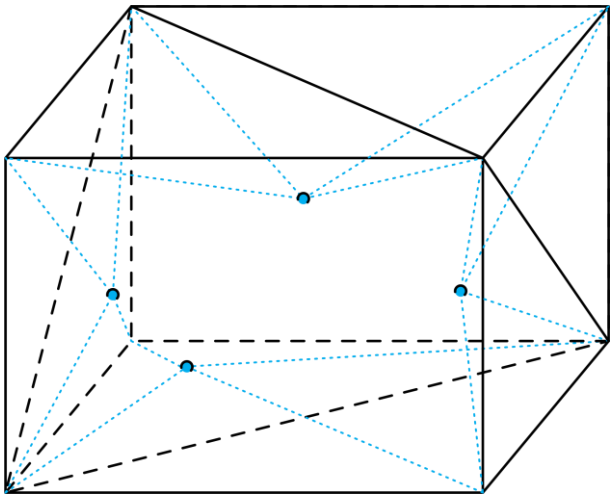
Pro tetrahedronizaci je možné použít jakoukoliv metodu tetrahedronizace. V případě této práce bude využita Delaunayova metoda pomocí inkrementálního vkládání. Prvním krokem metody inkrementálního vkládání je nalezení min max boxu množiny tetrahedronizovaných bodů a v dalším kroku vytvoření počátečního velkého tetrahedronu. V tomto případě je ovšem znám obalový kvádr a i osm bodů, které ho definují. Této vlastnosti je možné využít a vytvořit tak počáteční tetrahedronizaci, do které budou poté postupně vkládány jednotlivé body. Z množiny bodů je nutné vybrat jeden náhodný bod, který leží uvnitř obalového kvádru, tudíž není jeho vrcholem. Každou stěnu kvádru je možné rozdělit na dva trojúhelníky. Každý trojúhelník je možné spojit s vybraným bodem uvnitř kvádru a vytvořit tak vždy jeden nový tetrahedron (viz Obr. 5.1). V dalším kroku je nutné legalizovat všechny tetrahedrony a zajistit tak splnění Delaunayovského kritéria.



Obr. 5.1: Vložení prvního bodu do obalového kvádru.

Nyní je již možné pokračovat v běžném inkrementálním vkládání bodů do tetrahedronizace. Po vložení všech bodů do tetrahedronizace se již neprování odstraňování počátečního tetrahedronu, neboť body definující obalový kvádr jsou součástí tetrahedronizace.

Po dokončení tetrahedronizace gridu je nutné nalézt některé body, které budou využívány pro budoucí spojování tetrahedronizací. Takových bodů je v každém gridu šest (stejný počet jako počet stěn kvádru). Pro každou stěnu je nutné nalézt bod, který není vrcholem kvádru a je obsažen v tetrahedronu, jehož jedna stěna je totožná s danou stěnou (viz Obr. 5.2). Nalezených šest bodů nemusí být nutně šest různých bodů. V krajním případě se může stát, že všech šest bodů bude totožných (to v případě, že uvnitř gridu byl pouze jeden bod).

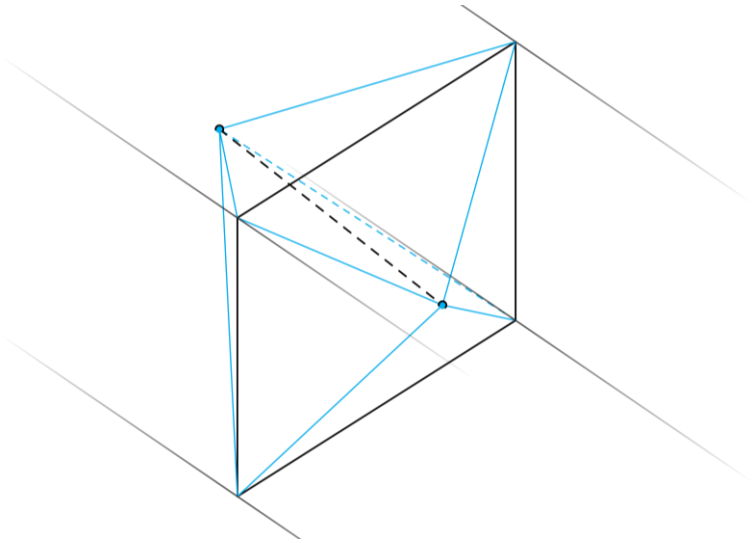


Obr. 5.2: Spojovací body (nezobrazeny body pro přední a zadní stěnu kvádru).

## 5.3 Spojení jednotlivých množin

Jednotlivé množiny bodů byly nezávisle tetrahedronizovány a v dalším kroku je nutné jednotlivé tetrahedronizace spojit do jednoho celku. Pro spojování se využijí body, které byly nalezeny po dokončení každé tetrahedronizace (viz Obr. 5.2).

Pro spojení tetrahedronizací je nutné projít všechny stěny kvádrů, které jsou obsaženy ve dvou gridech. To znamená všechny stěny, kromě stěn na povrchu min max boxu všech bodů. Procházení těchto stěn je opět možné provádět paralelně. Dva gridy jsou vždy spojeny přes společnou stěnu pomocí již dříve nalezených bodů a to tak, že se mezi těmito gridy vytvoří čtyři nové tetrahedrony (viz Obr. 5.3). Původní tetrahedrony, které obsahovali již dříve nalezený bod a jejich jedna stěna je totožná se stěnou mezi aktuálními dvěma gridy jsou z tetrahedronizace odebrány.



Obr. 5.3: Spojení tetrahedronizací se společnou stěnou.

V případě, že by bylo možné ponechat všechny navíc přidané body ve výsledné tetrahedronizaci, tak je v tomto okamžiku tetrahedronizace dokončena. V opačném případě je nutné tyto body postupně odebrat. Způsob, jakým jsou body odebírány je popsán v následujících kapitolách.

### 5.3.1 Odebrání vnitřních řídících bodů z tetrahedronizace

Problém odstranění bodu z tetrahedronizace se dá řešit pomocí dvou různých způsobů. První možností je odebrat z tetrahedronizace všechny tetrahedrony, které obsahují odstraňovaný bod. Tímto způsobem vznikne v tetrahedronizaci díra, kterou je nutné znovu retetrahedronizovat. Druhou možností je postupným posouvání přemístit odstraňovaný bod k jinému bodu v tetrahedronizaci. Posouvání je nutné provádět po určitých krocích a po každém kroku nějakým způsobem upravit tetrahedronizaci.

Přístup odstraňování bodu z tetrahadronizace, který bude využíván pro odebrání vnitřního řídícího bodu, bude postupné posouvání bodu k jinému bodu

v tetrahedronizaci. Je totiž výhodnější využít strukturu tetrahedronů a pouze jí upravovat, než vytvářet všechny tetrahedrony zcela od začátku.

Místo, kam se bude odstraňovaný bod pohybovat, bude nejbližší bod k odstraňovanému bodu. Čím je totiž vzdálenost přesunutí budu kratší, tím by měl být počet nutných úprav tetrahedronizace nižší.

Hlavní otázkou, kterou je nutné zodpovědět, je velikost jednotlivých kroků posouvání bodu. Neboli jak daleko je možné posunout vrchol v určitém směru, aby v tetrahedronizaci nevznikly protínající se tetrahedrony. Pokud se dva tetrahedrony protnou, tak to znamená, že jeden tetrahedron při posouvání bodu změnil svou orientaci. Orientaci tetrahedronu s vrcholy $A$, $B$, $C$ a $D$ lze zapsat následovně

$$\begin{aligned} V_0^{(i)} &= \begin{vmatrix} A_x^{(i)} & A_y^{(i)} & A_z^{(i)} & 1 \\ B_x^{(i)} & B_y^{(i)} & B_z^{(i)} & 1 \\ C_x^{(i)} & C_y^{(i)} & C_z^{(i)} & 1 \\ D_x^{(i)} & D_y^{(i)} & D_z^{(i)} & 1 \end{vmatrix} \\ &= \begin{vmatrix} A_x^{(i)} - B_x^{(i)} & B_x^{(i)} - C_x^{(i)} & B_x^{(i)} - D_x^{(i)} \\ A_y^{(i)} - B_y^{(i)} & B_y^{(i)} - C_y^{(i)} & B_y^{(i)} - D_y^{(i)} \\ A_z^{(i)} - B_z^{(i)} & B_z^{(i)} - C_z^{(i)} & B_z^{(i)} - D_z^{(i)} \end{vmatrix}, \end{aligned} \tag{5.5}$$

kde $V_0^{(i)}$ je orientace $i$-tého vrcholu. Nyní lze zvolit, že vrchol tetrahedronu, který se bude pohybovat je například vrchol $A$. Poté lze novou pozici vrcholu zapsat pomocí parametrického předpisu

$$A' = A + \lambda_i \Delta, \tag{5.6}$$

kde $\lambda \in \mathbb{R}$, $\Delta = (\Delta_x, \Delta_y, \Delta_z)$ a $\Delta$ je směrový vektor posouvání odstraňovaného bodu. Finální pozice přemísťovaného bodu se získá dosazením hodnoty 1 za parametr $\lambda$. Následně lze přepsat vzorec (5.6) pomocí využití vzorce (5.5) jako

$$V_1^{(i)} = V_0^{(i)} + \lambda_i \begin{vmatrix} \Delta_x & B_x^{(i)} - C_x^{(i)} & B_x^{(i)} - D_x^{(i)} \\ \Delta_y & B_y^{(i)} - C_y^{(i)} & B_y^{(i)} - D_y^{(i)} \\ \Delta_z & B_z^{(i)} - C_z^{(i)} & B_z^{(i)} - D_z^{(i)} \end{vmatrix}. \tag{5.7}$$

Maximální pozice, kam až lze odstraňovaný bod přesunout je taková, kdy hodnota $V_1^{(i)}$ dosáhne hodnoty nula. To znamená, že se všechny čtyři body definující tetrahedron ocitnou v jedné rovině a výsledek orientačního testu (5.5) a i (5.7) bude roven nule. Díky tomuto poznatku lze vypočíst pro každý tetrahedron hraniční hodnotu parametru $\lambda_i$, kam až se může bod maximálně posunout bez nutnosti tetrahedron nijak transformovat. Výpočet hodnot $\lambda_i$ se vypočte podle vzorce

$$\lambda_i = \frac{-V_0^{(i)}}{\begin{vmatrix} \Delta_x & B_x^{(i)} - C_x^{(i)} & B_x^{(i)} - D_x^{(i)} \\ \Delta_y & B_y^{(i)} - C_y^{(i)} & B_y^{(i)} - D_y^{(i)} \\ \Delta_z & B_z^{(i)} - C_z^{(i)} & B_z^{(i)} - D_z^{(i)} \end{vmatrix}}. \tag{5.8}$$

Hodnoty parametru $\lambda_i$ je nutné vypočíst pro všechny tetrahedrony, které obsahují jako jeden ze svých vrcholů odstraňovaný bod $A$. Zajímavé hodnoty parametru $\lambda_i$ jsou poze ty z intervalu $\lambda_i \in (0; 1\rangle$. Tyto hodnoty je nutné seřadit vzestupně a tím tak získat pozice, do kterých se postupně bude odstraňovaný bod přemisťovat.

Během každého posunutí bodu $A$ do pozice $A' = A + \lambda_i \Delta$ se musí transformovat tetrahedron s indexem $i$. Tento tetrahedron má trojúhelník, který neobsahuje bod $A$ a je k němu nutné nalézt jeden ze tří sousedních trojúhelníků na povrchu díry, který s ním má společnou hranu a zárovň mohou společně vytvořit nový tetrahedron. Nově vytvořený tetrahedron musí vyplnit pomyslnou díru při odstranění bodu $A$. Je tedy nutné kontrolovat správnou orientaci tetrahedronu, aby neprotnul žádný sousední tetrahedron. Tento nově vytvořený tetrahedron je vložen do tetrahedronizace a zároveň jsou odstraněny oba tetrahedrony, jejichž trojúhelníky byly použity k jeho tvorbě. Dále je potřeba do tetrahedronizace přidat dva nové tetrahedrony, které budou obsahovat bod $A$ a dva nově vytvořené trojúhelníky, které v tetrahedronizaci předtím nebyly. Pro tyto dva tetrahedrony je nutné vypočíst hodnotu parametru $\lambda_i$ a zařadit ji mezi seřazené hodnoty. Zajímavé hodnoty parametru $\lambda_i$ jsou nyní již jenom hodnoty z intervalu $\lambda_i \in \langle \lambda_{poslední}; 1\rangle$, $\lambda_{poslední}$ je poslední použitá hodnota parametru $\lambda_i$ pro posunutí odstraňovaného bodu.

Po zpracování všech zajímavých hodnot parametru $\lambda_i$ je nutné z tetrahedronizace odstranit všechny tetrahedrony, které obsahují jako vrchol bod $A$ a zároveň $A'$. V dalším kroku je nutné přepstat v tetrahedronech všechny vrcholy A na vrchol $A'$. V tomto okamžiku je bod $A$ úspěšně odstraněn z tetrahedronizace.

## 5.4 Odebrání přidaných bodů

V případech, kdy grid neobsahoval žádný vnitřní bod z množiny vstupních bodů, byl uvnitř tohoto gridu vygenerován nový náhodný bod. V tomto okamžiku je nutné tento bod odstranit. Odstranění bodu probíhá pomocí stejného algoritmu jako odstraňování vnitřních řídících bodů z tetrahedronizace (viz kap. 5.3.1).

## 5.5 Tvorba konvexní obálky

Každá tetrahedronizace musí splňovat podmínku, že povrch tetrahedronizace je zároveň konvexní obálkou všech bodů tetrahedronizace. Vytvoření konvexní obálky proběhne postupným odebráním všech přidaných řídících bodů, které jsou na povrchu min max boxu všech vstupních bodů. Po odebrání těchto bodů zůstanou

v tetrahedronizaci pouze body, které byly ve vstupní množině bodů pro tetrahedronizaci.

Pro odebírání bodů je využit algoritmus pro odstranění bodu uvnitř tetraheronizace (viz kap. 5.3.1) s drobnými modifikacemi. Změny se týkají parametrů $\lambda_i$ pro tetrahedrony získané ze vzorce (5.8). V tomto případě jsou vždy zajímavé všechny hodnoty z intervalu $\lambda_i \in (0; \infty)$. Stále se postupně vybírá nejnižší možná hodnota parametru $\lambda_i$. Další změnou je, že pokud nelze vytvořit nový tetrahedron pomocí žádných ze tří sousedních trojúhelníků, je pouze hodnota parametru $\lambda_i$ odebrána ze seznamu zajímavých hodnot.

V okamžiku, kdy je seznam zajímavých hodnot parametru $\lambda_i$ prázdný, je nutné z tetrahedronizace odstranit všechny tetrahedrony, které obsahují jako vrchol bod $A$ a zároveň $A'$. V dalším kroku je nutné přepstat v tetrahedronech všechny vrcholy $A$ na vrchol $A'$. V tomto okamžiku je bod $A$ úspěšně odstraněn z tetrahedronizace.

Po odstranění všech přidaných bodů obsahuje tetrahedronizace již pouze vstupní body pro tetrahedronizaci. Algoritmus pro tvorbu tetrahedronizace tedy v tomto okamžiku končí.

# 6 Implementace

V předchozích kapitolách byly prezentovány nové algoritmy pro triangulaci množiny bodů v $E^2$ a tetrahedronizaci množiny bodů v $E^3$. Tyto algoritmy byly implementovány, aby bylo možné ověřit jejich správnost a jejich vlastnosti. V následujících kapitolách budou popsány jednotlivé implementace, které byly vytvořeny.

## 6.1 Paralelní triangulace v E²

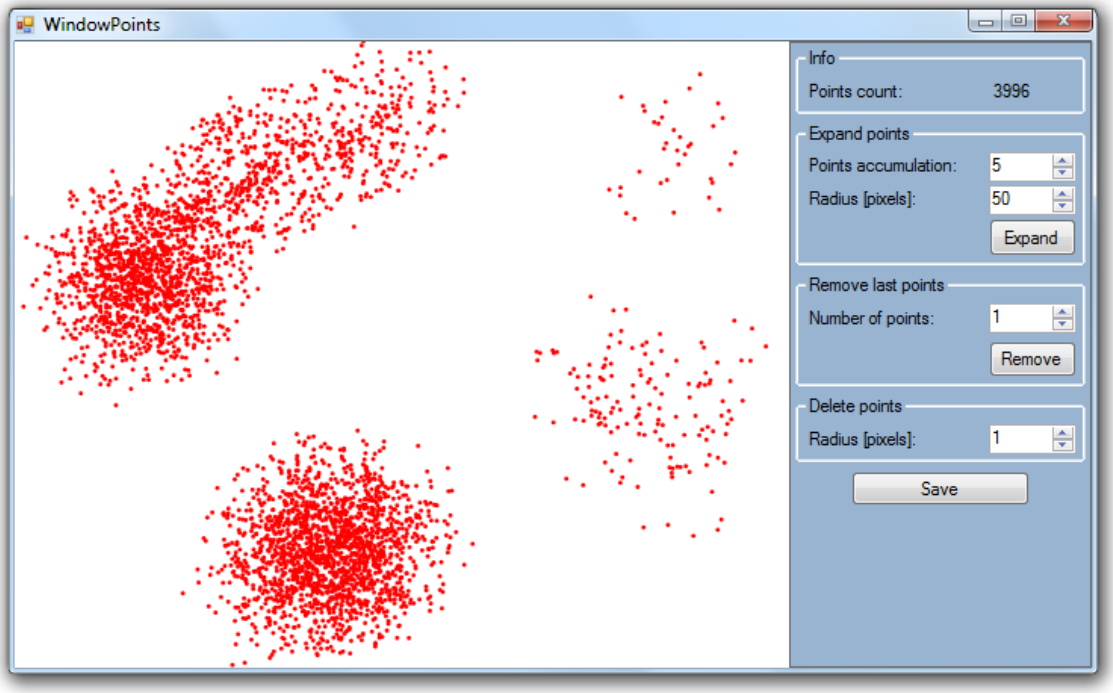
Implementace paralelní triangulace probíhala ve více fázích. V první fázi byla provedena implementace v programovacím jazyce C# a to pouze pro účel ověření správnosti navrženého algoritmu. V druhé fázi byla provedena implementace na GPU pomocí CUDA [NVi], kterou je možné přeložit a spustit na GPU nebo pomocí podmíněného překladu i na CPU.

Pro triangulaci bodů v jednom gridu byla ve všech níže popsaných implementacích využita implementace triangulace, jejíž očekávaná složitost je $O(N^2)$.

### 6.1.1 C# verze triangulace

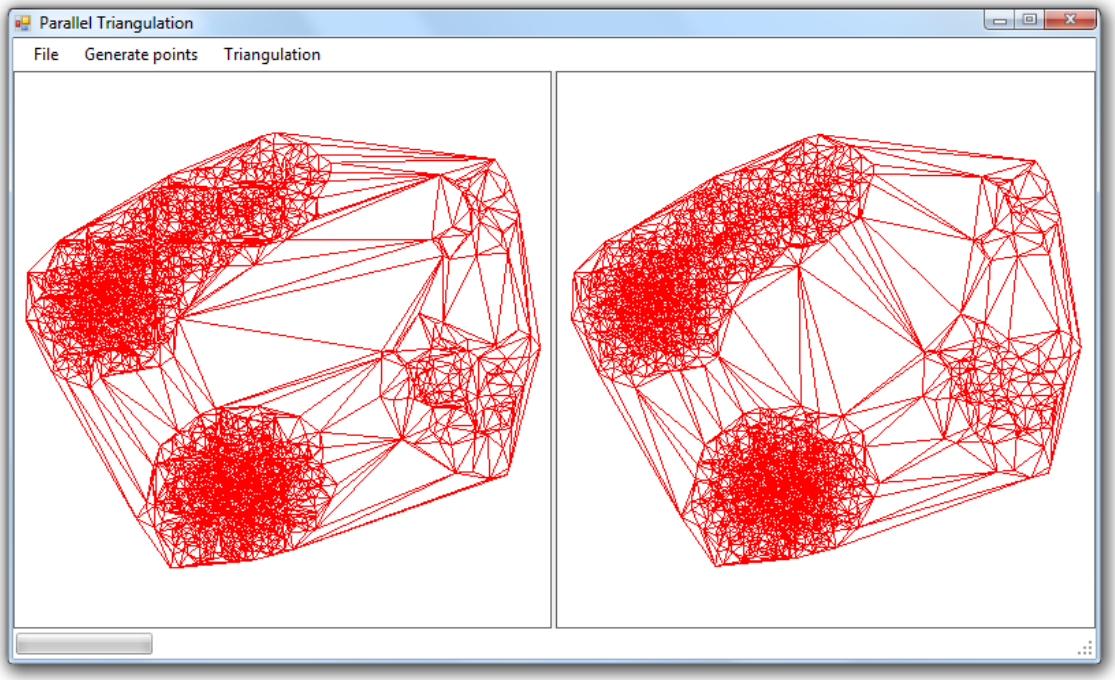
Nejprve bylo nutné otestovat, zda navržený algoritmus funguje správně a lze pomocí něj triangulizovat různě velké množiny bodů s různým typem rozložení. Pro tuto implementaci byl vybrán programovací jazyk C# a grafická knihovna SlimDX [SDX] pro zobrazení výsledné triangulace.

Pomocí implementovaného programu je možné automaticky vygenerovat body s náhodným a gausovským rozložením a ty poté pomocí paralelní metody triangulizovat. Dále je možné využít GUI aplikace a ručně vygenerovat body pro triangulaci (viz Obr. 6.1). Nejprve se pomocí myši zvolí několik základních bodů, kolem kterých je poté možné automaticky vygenerovat náhodné body s gausovským rozložením.




Obr. 6.1: Generování bodů.

Vygenerované body lze uložit na disk a později triangulizovat pomocí navržené paralelní metody i pomocí Delaunayovy triangulace. Obě triangulace se poté zobrazují vedle sebe, aby je bylo možné navzájem porovnat (viz Obr. 6.2). Delaunayova triangulace se ovšem počítá pouze v případě, že počet vstupních bodů pro triangulaci je nízký (maximálně řády tisíců), neboť očekávaná algoritmická složitost implementované metody je $O(N^2)$.




Obr. 6.2: Triangulace bodů. Vlevo je triangulace pomocí paralelní metody, vpravo je Delaunayova triangulace.

Na (Obr. 6.2) je možné pozorovat, že se obě triangulace od sebe výrazně liší hlavně ve svých dlouhých hranách. V paralelní metodě jsou tyto hrany (trojúhelníky) vytvářeny při spojování triangulací jednotlivých gridů a při odebírání řídících bodů, ale již není kontrolováno Delaunayovo kritérium.

### 6.1.2 GPU triangulace

Navržená paralelní triangulace se zdá být vhodná pro GPU. Počet nezávislých výpočtů, které lze provádět paralelně je velmi mnoho (pro $10^5$ bodů je počet nezávislých výpočtů v řádech tisíců). Pro implementaci triangulace na GPU byla využita CUDA spolu s programovacím jazykem C++.

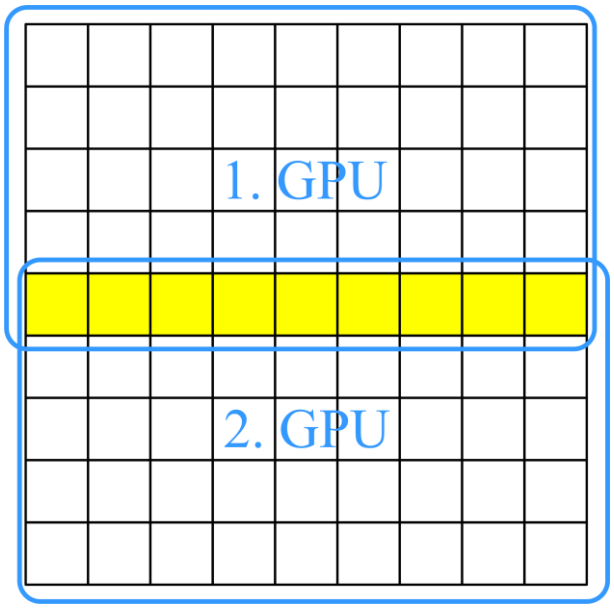
CUDA ( = Compute Unified Device Architecture) [NVi] je paralelní výpočetní platforma vytvořená společností NVIDIA. CUDA je implementována pomocí grafických procesorů od NVIDIA. CUDA dává programátorům přístup k virtuálnímu setu instrukcí a paměti na GPU.

Na GPU jsou počítány pouze ty části triangulace, které jsou paralelizovány. První částí je triangulace jednotlivých gridů a druhou částí je spojení triangulací z jednotlivých gridů. Před spuštěním výpočtu na GPU je nutné alokovat paměť na GPU a nakopírovat potřebná data pro výpočet. Po skončení výpočtu na GPU je nutné získaná data zkopírovat zpět z GPU paměti do paměti RAM, aby mohla být triangulace dokončena pomocí CPU.

Při využívání GPU je nutné alokovat veškerou potřebnou paměť předem a při samotném výpočtu na GPU nelze paměť dynamicky alokovat. Z tohoto důvodu není možné používat na GPU žádné dynamické struktury. Velikost používaných struktur při výpočtu musí být určena již při kompilaci programu. Zároveň maximální počet vytvořených trojúhelníků v každém gridu musí být stanoven před samotným výpočtem triangulací jednotlivých gridů. Podmínku maximálního počtu trojúhelníků pro jeden grid lze splnit jednoduše, neboť z počtu triangulizovaných bodů v gridu lze vypočíst maximální možný počet v budoucnu vytvořených trojúhelníků.

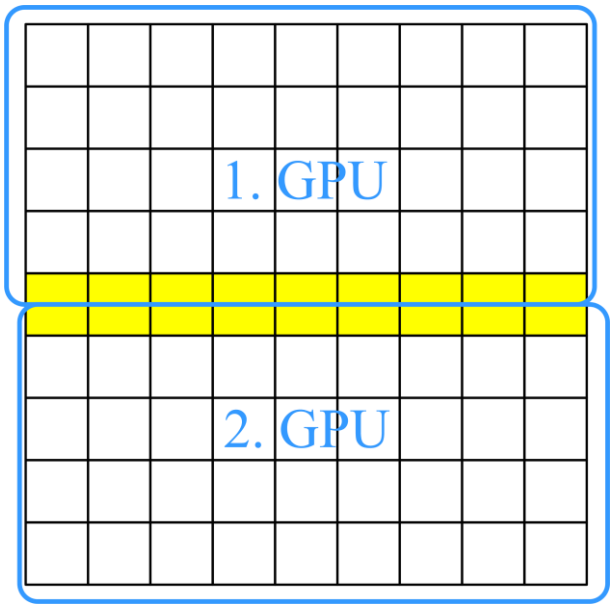
Zásadním problémem je ovšem, že velikost používaných struktur při výpočtu musí být určena již při kompilaci programu. Používaná konstanta velikosti je tedy nastavena pro určitý maximální počet bodů na grid. Pokud ovšem nastane situace, kdy by bylo potřeba strukturu realokovat (zvětšit její velikost), nebude to možné. Z tohoto důvodu je možné pomocí GPU triangulizovat pouze body s rovnoměrným rozložením, neboť po rozdělení bodů do gridů bude v každém gridu přibližně stejný počet bodů.

Implementovaná paralelní triangulace na GPU je schopna využívat i více GPU, pokud jsou k dispozici na daném počítači. V takovém případě je nutné mezi GPU práci rozdělit. V první fázi je prováděna triangulace jednotlivých gridů a je nutné, aby se jeden řádek gridů triangulizoval na dvou GPU (viz Obr. 6.3).




Obr. 6.3: Rozdělení gridů pro triangulaci mezi dvě GPU. Zvýrazněné gridy jsou triangulizovány na obou GPU.

Během spojování triangulací je využito, že jeden řádek gridů byl triangulizován na dvou GPU a tudíž není nutné mezi GPU kopírovat žádná data. Na každém GPU se triangulizuje vzniklá hvězdice mezi každými čtyřmi sousedními gridy (viz Obr. 6.4).




Obr. 6.4: Spojení triangulací gridů (triangulizuje se vždy hvězdice mezi 4 gridy).

Po dokončení triangulace na všech GPU se ze všech GPU zkopírují vzniklé triangulace do paměti RAM. Pomocí CPU jsou poté dokončeny zbylé fáze triangulace, které již nejsou vykonávány paralelně.

### 6.1.3 CPU triangulace

Triangulaci, která byla implementována pro běh na GPU (C++ a CUDA), je možné pomocí podmíněného překladu přeložit i pro CPU (C++). Rozdíl ve vykonávaném kódu pomocí CPU a GPU verze triangulace není téměř žádný. Díky této skutečnosti je možné porovnat dva téměř totožně napsané algoritmy, kde jeden běží s využitím GPU a druhý pouze na CPU.

Pro paralelizaci výpočtu bylo využito OpenMP [OMP]. OpenMP je API ( = Application Program Interfeace), které podporuje paralelní programování se sdílenou pamětí v C/C++ a Fortran. Skládá se z direktiv pro překladač, knihovny a několika globálních proměnných, které ovlivňují běh programu.

## 6.2 Paralelní tetrahedronizace v $E^3$

Programovací jazyk, který byl použit pro implementaci navržené metody tetrahedronizace, je C++. Pro paralelizaci výpočtu bylo, jako v případě implementace triangulace na CPU, využito OpenMP.

Části tetrahedronizace, které se mohou vykonávat paralelně, jsou tetrahedronizace jednotlivých gridů a poté spojování tetrahedronizací jednotlivých gridů. Při tetrahedronizaci jednotlivých gridů je možné provádět všechny tetrahedronizace nezávisle a tudíž při paralelním běhu není nutná žádná komunikace mezi vlákny.

Při spojování tetrahedronizací jednotlivých gridů se vždy postupně přesouvá jeden řídící bod, aby se nakonec mohl odstranit. Při přesouvání se mohou měnit tetrahedrony ve všech osmi okolních gridech (gridy, které obsahují odebíraný bod jako svůj gridový bod). Tudíž není možné provádět spojování gridů najednou bez nutnosti vnitřní komunikace. Je ovšem ale možné gridy rozdělit do disjunktních množin vždy o osmi gridech a poté v každé množině osmi gridů odstranit řídící gridový bod, který leží ve středu daných osmi gridů. Disjunktní množiny po osmi gridech lze vytvářet osmi různými způsoby tak, že nakonec budou postupně odebrány všechny vnitřní řídící body. Odebírání řídících bodů v takovém případě může v rámci každé disjunktní množiny osmi gridů probíhat paralelně bez nutnosti vnitřní komunikace mezi vlákny.

# 7 Testování

Testy triangulace a tetrahedronizace probíhaly na školním počítači, jehož sestava je popsána níže:

- CPU: Intel(R) Core(TM) i7 920 (4 × 2,67GHz) s 8 HyperThreads
- paměť: 12 GB RAM
- GPU: 2 GeForce GTX 295
    - 30 multiprocesorů × 8 CUDA Cores na jeden multiprocesor (1,38GHz)
    - paměť: 896MB (1,05GHz)
- operační systém: Microsoft Windows 7 64bit

## 7.1 Paralelní triangulace v $E^2$

Při testování triangulace byla v první fázi vstupní množina bodů vždy generována tak, že body se nacházely v prostoru $\langle -1; 1\rangle \times \langle -1; 1\rangle$. Body byly generovány s náhodným rozložením. Po otestování triangulace na bodech s náhodným rozložením byly provedeny testy na reálných datech (viz kap. 7.1.5).

### 7.1.1 Počet bodů na grid

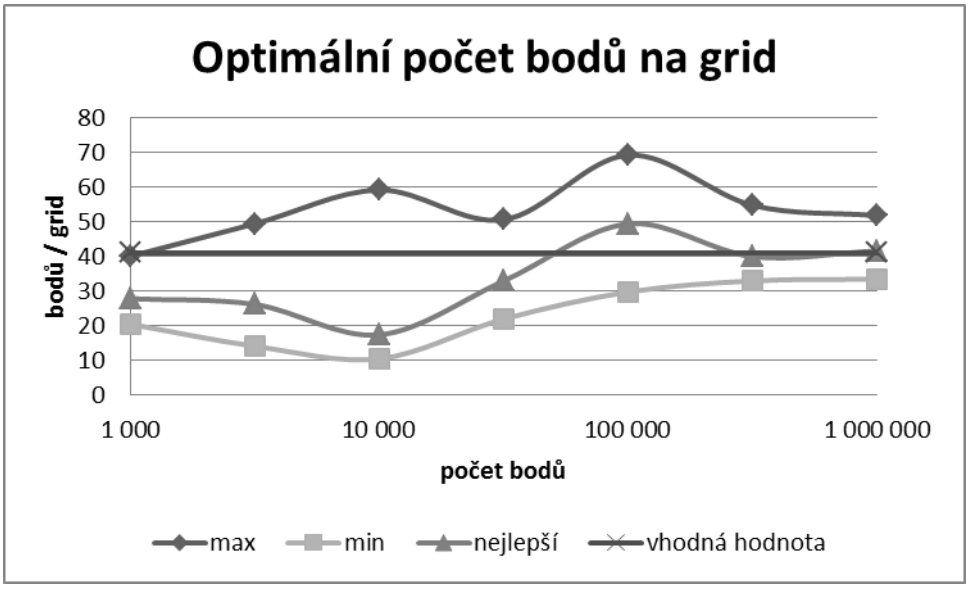
Jedním z parametrů, na kterém závisí doba triangulace, je počet dělení v každé ose, neboli také průměrný počet bodů na jeden grid. Pokud bude totiž v jenom gridu příliš mnoho bodů, bude triangulace trvat dlouhý čas, ovšem spojení triangulací a tvorba konvexní obálky bude rychlá. Opačný případ je, když počet bodů na jeden grid je příliš malý. V tom případě je triangulace gridů rychlá, ovšem spojení jednotlivých triangulací a tvorba konvexní obálky bude trvat dlouhou dobu. Z tohoto důvodu je nutné nalézt optimální hodnotu pro počet bodů na jeden grid.

#### 7.1.1.1 Testování na CPU

Triangulace probíhala s využitím osmy vláken, což je maximum vláken, které mohou běžet na daném procesoru paralelně (s využitím metody Hyperthreading). Testování probíhalo na náhodně rozložených datech o různých velikostech. Velikosti vstupních množin byly voleny jako násobky čísla $\sqrt{10}$ a nejnižší počet testovaných bodů byl $1000$.

Pro každou množinu bodů byly postupně zkoušeny různé hodnoty počtu bodů na jeden grid. Časová náročnost algoritmu pro různé počty vstupních bodů v závislosti na počtu bodů v gridu je zobrazena v grafech (Graf 11.1 až Graf 11.7). Průběh všech grafů je přibližně stejný. Nejprve se časová náročnost triangulace snižuje se zvyšujícím se počtem bodů na grid. V určitém momentu se situace obrátí a časová náročnost triangulace se začne zvětšovat. V každém grafu je tedy možné nalézt minimum a

přibližný interval, ve kterém jsou všechny časy přibližně shodné a zároveň minimální. Tyto hodnoty jsou zobrazeny v grafu (Graf 7.1).



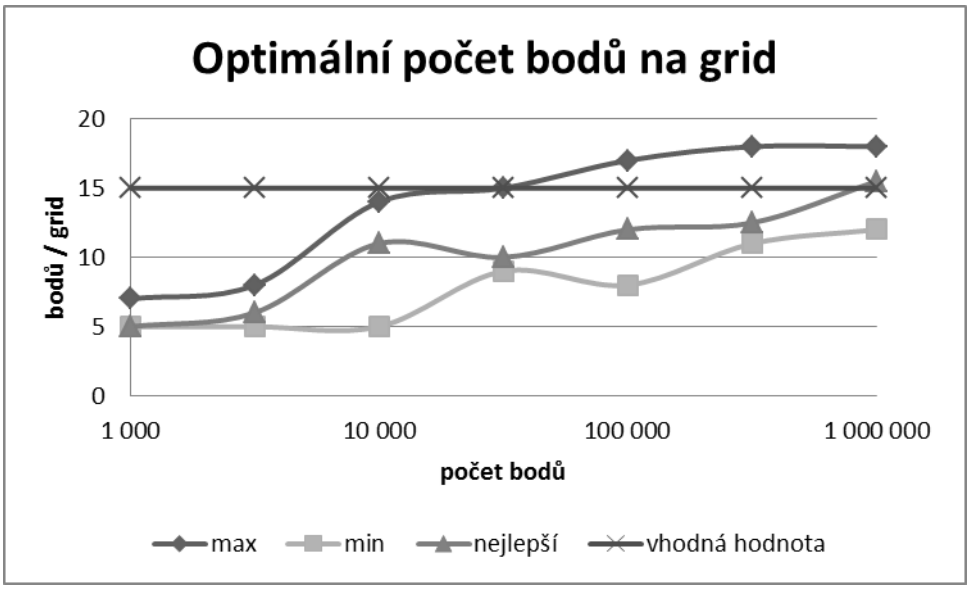

Graf 7.1: Souhrnné zobrazení optimálního počtu bodů na grid.

V grafu (Graf 7.1) představuje plocha mezi křivkami $min$ a $max$ všechny možné vhodné kombinace počtu vstupních bodů a počtu bodů na jeden grid. Dále je zde zobrazena křivka, kdy bylo dosaženo nejnižší časové náročnosti triangulace.

Optimální hodnota počtu bodů na grid může být zvolena konstantní a to velikosti 41 bodů na grid. Pomocí této hodnoty bude tedy vždy vypočten počet dělení v každé ose.

#### 7.1.1.2 Testování na GPU

Testování optimálního počtu bodů na grid probíhalo stejným způsobem, jakým probíhalo testování na CPU. Výsledné grafy znázorňující časovou náročnost triangulace na počtu bodů na grid jsou zobrazeny v grafech (Graf 11.8 až Graf 11.14). Výsledný souhrn z těch grafů je zobrazen na následujícím grafu.




Graf 7.2: Souhrnné zobrazení optimálního počtu bodů na grid.

Z grafu (Graf 7.2) je vidět, že vhodný počet bodů na grid je mnohem nižší než při testování na CPU. Důvod je pomalejší výpočet triangulace na GPU. Pokud by byla frekvence grafického procesoru vyšší nebo pokud by bylo možné využít větší množství mikroprocesorů, byl by výpočet rychlejší a tím pádem by se i zvýšil optimální počet bodů na grid.

Jako optimální hodnota počtu bodů na grid byl zvolen konstantní počet 15 bodů na jeden grid. Pomocí této hodnoty je opět možné vypočíst příslušný počet dělení v každé ose. Pro malé počty vstupních bodů je tato zvolená vhodná hodnota počtu bodů na grid mnohem vyšší, než nejlepší hodnota. Celková doba triangulace je ovšem pro malé počty bodů velmi nízká a tudíž je časové navýšení doby triangulace téměř zanedbatelné.

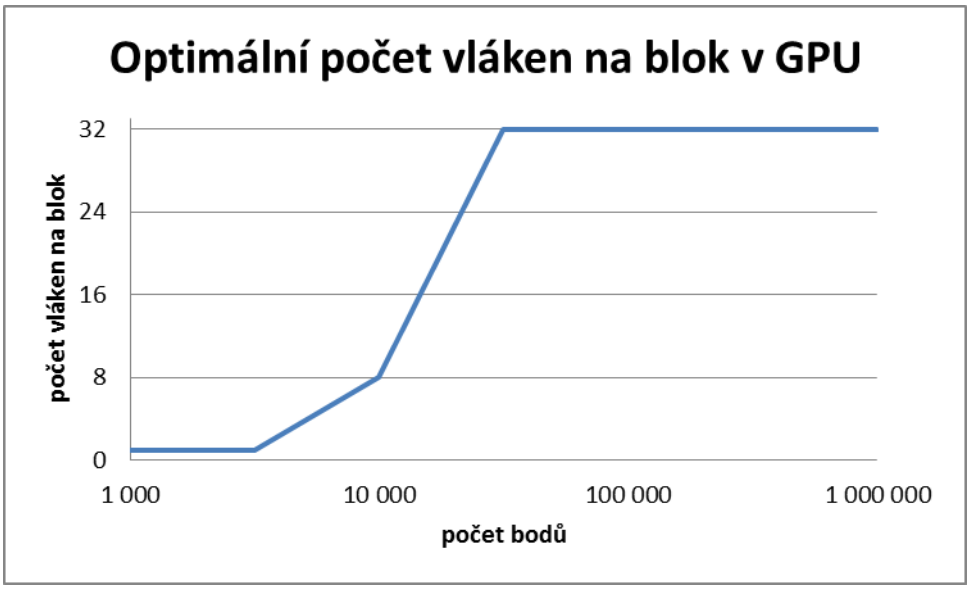
### 7.1.2 Počet vláken na blok v CUDA (GPU)

Každý CUDA program musí při spuštění kernelu nastavit počet vláken na jeden blok. Nejvhodnější počet vláken na grid je vhodné otestovat a získat tak optimální hodnotu, která bude využívána při spouštění triangulace na GPU. Pro různé počty vstupních bodů byl kernel postupně spouštěn s různými hodnotami počtu vláken. Nejnižší hodnotou bylo pouze jedno vlákno a poté násobky čísla 8, neboť to je počet CUDA jader, který obsahuje jeden mikroprocesor na GPU. Měřen byl pouze čas doby běhu obou kernelů na jednom GPU. Výsledky měření jsou zobrazeny v grafech (Graf 11.15 až Graf 11.21). V tabulce (Tab. 7.1) je zobrazen vždy počet bloků spuštěných na GPU při daném počtu vláken a počtu bodů. Tučně vyznačené hodnoty počtu bloků jsou takové, při kterých byla naměřena nejnižší doba běhu.

| | | Počet vláken na blok v GPU | | | | | |
|---|---|---|---|---|---|---|---|
| Počet bodů | Počet gridů | 1 | 8 | 16 | 32 | 64 | 96 |
| 1 000 | 64 | **64** | 8 | 4 | 2 | 1 | 1 |
| 3 162 | 196 | **196** | 25 | 12 | 6 | 3 | 2 |
| 10 000 | 625 | 625 | **78** | 39 | 20 | 10 | 7 |
| 31 622 | 2 025 | 2 025 | 253 | 127 | **63** | 32 | 21 |
| 100 000 | 6 561 | 6 561 | 820 | 410 | **205** | 103 | 68 |
| 316 227 | 21 025 | 21 025 | 2 628 | 1 314 | **657** | 329 | 219 |
| 1 000 000 | 66 564 | 66 564 | 8 321 | 4 160 | **2 080** | 1 040 | 693 |

Tab. 7.1: Počet bloků spuštěných na GPU. Tučně vyznačeny nejlepší hodnoty.

Z tabulky (Tab. 7.1) je možné zjistit, že počet bloků, které jsou spuštěny na GPU musí být větší než 60, což je dvou násobek počtu mikroprocesorů $(2 \times 30 = 60)$. Počet vláken v závislosti na počtu bodů je znázorněn v grafu (Graf 7.3).




Graf 7.3: Počet vláken pro jeden blok v závislosti na počtu vstupních bodů.

### 7.1.3 Časová náročnost

Rychlost triangulace je jedním z důležitých aspektů kvality. V následujících kapitolách bude změřena časová náročnost triangulace na CPU a na GPU. Měření časových náročností lze rozdělit do několika částí, které jsou pro CPU i GPU shodné. V následujícím výčtu jsou popsány jednotlivé části:

- Rozdělení bodů: Vstupní množina bodů je rozdělena do jednotlivých gridů.
- Alokace paměti: Je alokována paměť pro uložení vytvořených trojúhelníků a další potřebná paměť.
- 1. Kernel: Triangulace jednotlivých gridů.
- 2. Kernel: Spojení triangulací jednotlivých gridů.

- Konvexní obálka: Vytvoření konvexní obálky pomocí postupného odebírání přidaných bodů kolem vstupní množiny bodů.
- Pole trojúhelníků: Spojení jednotlivých polí trojúhelníků do jednoho výsledného pole.

Výsledný čas triangulace je poté získán jako součet dob trvání jednotlivých částí triangulace.

#### 7.1.3.1 Testování na CPU

V této kapitole bude popsáno testování, které probíhalo na CPU s využitím 8 vláken. V následující tabulce jsou zobrazeny získané časy.

| | Doba trvání [ms] | | | | | | |
|---|---|---|---|---|---|---|---|
| Počet bodů | Rozdělení bodů | Alokace paměti | 1. Kernel na CPU | 2. Kernel na CPU | Konvexní obálka | Pole trojúhelníků | Celkem |
| 1 000 | 0,015 | 0,016 | 0,145 | 0,007 | 0,040 | 0,006 | 0,23 |
| 3 162 | 0,044 | 0,020 | 0,378 | 0,022 | 0,102 | 0,013 | 0,58 |
| 10 000 | 0,136 | 0,021 | 1,012 | 0,064 | 0,238 | 0,068 | 1,54 |
| 31 622 | 0,426 | 0,023 | 3,033 | 0,192 | 0,561 | 0,220 | 4,46 |
| 100 000 | 1,672 | 0,018 | 9,369 | 0,611 | 1,764 | 0,741 | 14,17 |
| 316 227 | 8,116 | 0,012 | 29,608 | 2,089 | 5,610 | 2,863 | 48,30 |
| 1 000 000 | 32,887 | 0,020 | 95,943 | 7,718 | 17,262 | 9,345 | 163,17 |
| 3 162 277 | 125,620 | 0,043 | 310,787 | 27,523 | 70,975 | 29,925 | 564,87 |
| 10 000 000 | 534,687 | 0,226 | 1 043,590 | 93,971 | 189,399 | 95,304 | 1 957,18 |

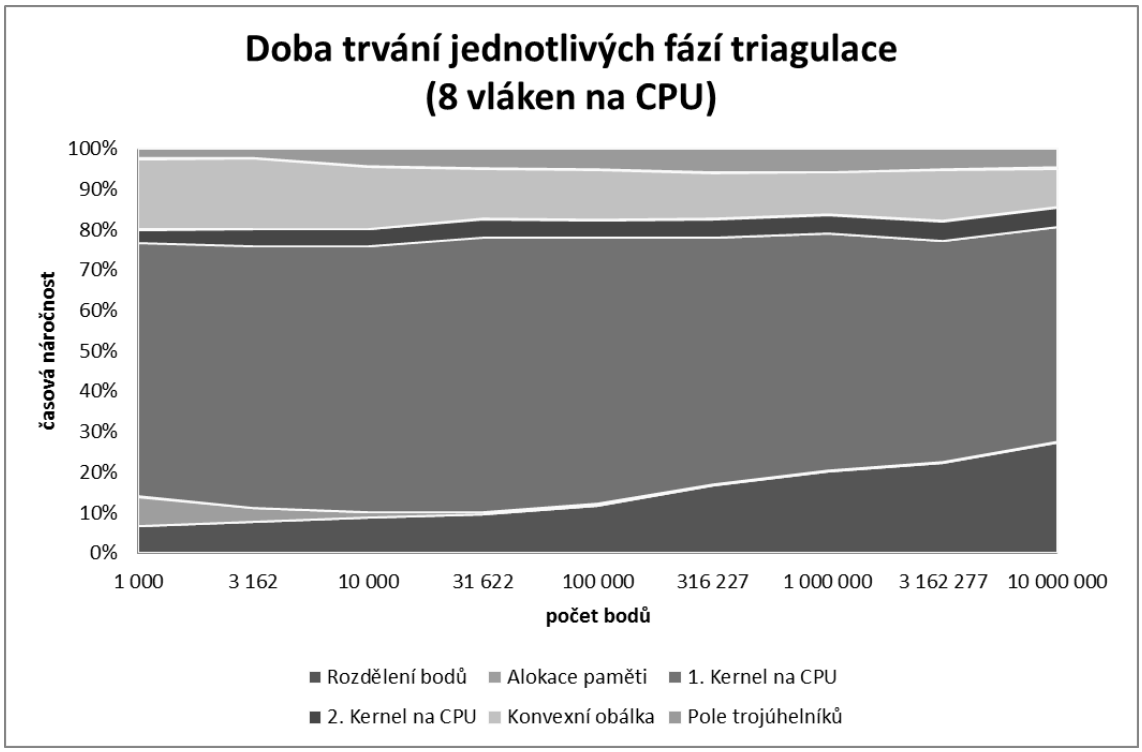
Tab. 7.2: Doba trvání triangulace v závislosti na počtu vstupních bodů.

Pomocí časů získaných měřením a shrnutých v tabulce (Tab. 7.2) je možné vypočíst procentuální náročnost jednotlivých fází triangulace vzhledem k celkové době triangulace. Tyto hodnoty jsou zobrazeny v následující tabulce.

| | Časová náročnost [%] | | | | | | |
|---|---|---|---|---|---|---|---|
| Počet bodů | Rozdělení bodů | Alokace paměti | 1. Kernel na CPU | 2. Kernel na CPU | Konvexní obálka | Pole trojúhelníků | Celkem |
| 1 000 | 6,7% | 7,2% | 63,0% | 3,2% | 17,5% | 2,5% | 100% |
| 3 162 | 7,7% | 3,4% | 65,2% | 3,8% | 17,6% | 2,3% | 100% |
| 10 000 | 8,9% | 1,4% | 65,7% | 4,2% | 15,5% | 4,4% | 100% |
| 31 622 | 9,6% | 0,5% | 68,1% | 4,3% | 12,6% | 4,9% | 100% |
| 100 000 | 11,8% | 0,1% | 66,1% | 4,3% | 12,4% | 5,2% | 100% |
| 316 227 | 16,8% | 0,0% | 61,3% | 4,3% | 11,6% | 5,9% | 100% |
| 1 000 000 | 20,2% | 0,0% | 58,8% | 4,7% | 10,6% | 5,7% | 100% |
| 3 162 277 | 22,2% | 0,0% | 55,0% | 4,9% | 12,6% | 5,3% | 100% |
| 10 000 000 | 27,3% | 0,0% | 53,3% | 4,8% | 9,7% | 4,9% | 100% |

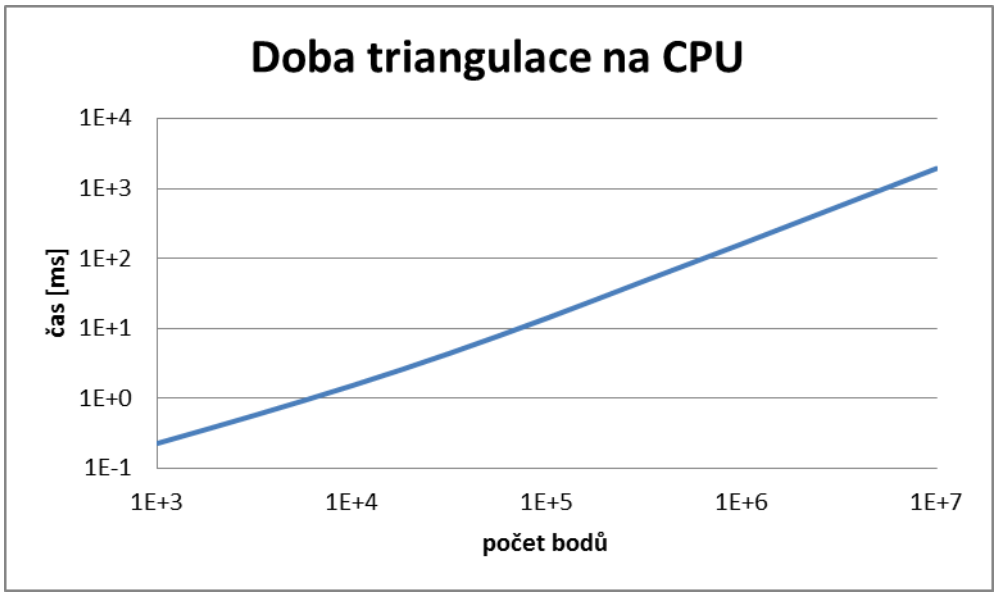
Tab. 7.3: Procentuální náročnost jednotlivých částí triangulace.

Z tabulky (Tab. 7.3) a grafu (Graf 7.4) je možné vyčíst, že nejdelší dobu trvá triangulace jednotlivých množin. Pokud by se měl snížit celkový čas triangulace, bylo by vhodné upravit právě část triangulace jednotlivých gridů. Nyní je pro tuto triangulaci využívána metoda, která má očekávanou časovou náročnost $O(N^2)$. Pokud by byla použita implementace triangulace s očekávanou složitostí $O(N \log N)$, byl by čas triangulace gridů snížen. S využitím hodnot z tabulky (Tab. 7.3) je možné vytvořit následující graf.




Graf 7.4: Procentuální náročnost jednotlivých částí triangulace.

Shrnutí časové náročnosti triangulace na CPU je zobrazeno v grafu (Graf 7.5). Průběh grafu je téměř lineární. To lze vysvětlit tím, že když se zvětší počet bodů na $k$ násobek, poté se také zvětší počet gridů na $k$ násobek, počet spojování gridů se také zvětší na $k$ násobek. Jelikož triangulace jednoho gridu a jedno spojení gridů trvá vždy stejnou dobu (počet bodů v gridu je vždy stejný) znamená to, že se celkový čas triangulace také zvýší na $k$ násobek původní hodnoty.




Graf 7.5: Časová náročnost triangulace.

### 7.1.3.2 Testování na GPU

V této kapitole bude popsáno testování triangulace na GPU. V první fázi byla časová náročnost triangulace testována pouze pomocí jednoho GPU. Výsledky z měření jsou zobrazeny v následující tabulce.

| | Doba trvání [ms] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Počet bodů | Rozdělení bodů | RAM -> GPU | 1. Kernel na GPU | 2. Kernel na GPU | GPU -> RAM | Konvexní obálka | Pole trojúhelníků | Celkem |
| 1 000 | 0,025 | 4,330 | 4,186 | 8,042 | 0,105 | 0,067 | 0,005 | 16,76 |
| 3 162 | 0,069 | 5,355 | 4,579 | 8,316 | 0,160 | 0,149 | 0,019 | 18,65 |
| 10 000 | 0,196 | 8,380 | 7,067 | 8,398 | 0,353 | 0,338 | 0,074 | 24,81 |
| 31 622 | 0,654 | 8,412 | 10,210 | 8,753 | 0,815 | 0,945 | 0,236 | 30,03 |
| 100 000 | 3,174 | 9,083 | 23,580 | 10,575 | 2,517 | 2,810 | 0,823 | 52,56 |
| 316 227 | 14,990 | 11,093 | 74,045 | 16,790 | 11,697 | 10,039 | 5,338 | 143,99 |
| 1 000 000 | 50,302 | 16,389 | 220,923 | 35,835 | 27,458 | 27,907 | 11,862 | 390,68 |

Tab. 7.4: Doba trvání triangulace na 1 GPU.

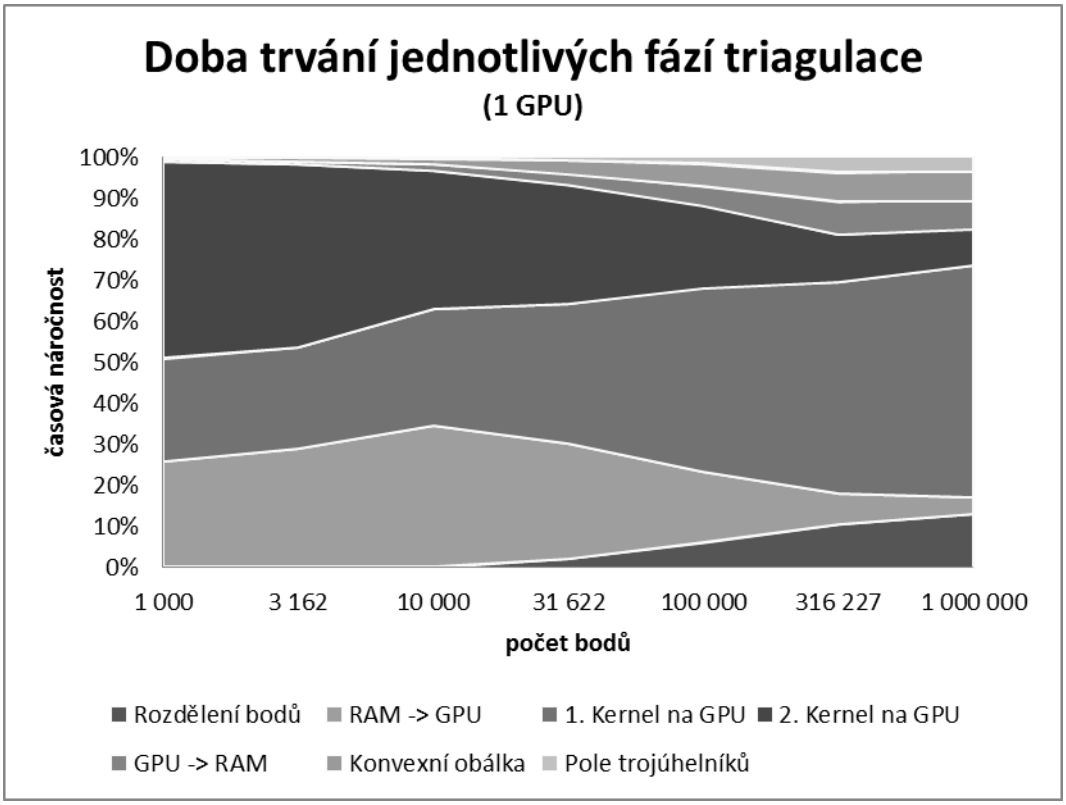
V tabulce (Tab. 7.4) jsou navíc oproti měření na CPU přidány položky RAM→GPU a GPU→RAM, které znamenají kopírování dat z RAM do paměti na grafické kartě a opačně.

Pomocí tabulky (Tab. 7.4) lze vytvořit tabulku (Tab. 7.5) a také graf (Graf 7.6), kde jsou zobrazeny procentuální náročnosti jednotlivých částí triangulace vzhledem k celkové době triangulace pro daný počet vstupních bodů.

| Počet bodů | Časová náročnost [%] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Rozdělení bodů | RAM -> GPU | 1. Kernel na GPU | 2. Kernel na GPU | GPU -> RAM | Konvexní obálka | Pole trojúhelníků | Celkem |
| 1 000 | 0,1% | 25,8% | 25,0% | 48,0% | 0,6% | 0,4% | 0,0% | 100% |
| 3 162 | 0,4% | 28,7% | 24,6% | 44,6% | 0,9% | 0,8% | 0,1% | 100% |
| 10 000 | 0,8% | 33,8% | 28,5% | 33,9% | 1,4% | 1,4% | 0,3% | 100% |
| 31 622 | 2,2% | 28,0% | 34,0% | 29,2% | 2,7% | 3,1% | 0,8% | 100% |
| 100 000 | 6,0% | 17,3% | 44,9% | 20,1% | 4,8% | 5,3% | 1,6% | 100% |
| 316 227 | 10,4% | 7,7% | 51,4% | 11,7% | 8,1% | 7,0% | 3,7% | 100% |
| 1 000 000 | 12,9% | 4,2% | 56,5% | 9,2% | 7,0% | 7,1% | 3,0% | 100% |

Tab. 7.5: Časová náročnost jednotlivých fází triangulace na 1 GPU.

Z tabulky (Tab. 7.5) a také grafu (Graf 7.6) je možné vyčíst následující vlastnosti. Při malém počtu vstupních bodů je kopírování (a zároveň i alokace paměti) dat z RAM do GPU paměti časově náročné vzhledem k celkové době triangulace. Dále při malém počtu bodů je pomalý druhý kernel počítaný na GPU. To je nejspíše zapříčiněno tím, že nějakou krátkou dobu trvá, než se vůbec kernel začne počítat. Na GPU se totiž musí provést i ostatní úlohy operačního systému po ukončení prvního kernelu.




Graf 7.6: Časová náročnost jednotlivých fází triangulace na 1 GPU.

Při druhém testování časové náročnosti triangulace na GPU byly využity dvě GPU. Provedené testy se shodují s testy, které byly vykonány pomocí pouze jednoho GPU. Získané naměřené hodnoty jsou zobrazeny v následující tabulce.

| | Doba trvání [ms] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Počet bodů | Rozdělení bodů | RAM -> GPU | 1. Kernel na GPU | 2. Kernel na GPU | GPU -> RAM | Konvexní obálka | Pole trojúhelníků | Celkem |
| 1 000 | 0,025 | 3,596 | 2,497 | 4,212 | 0,095 | 0,067 | 0,005 | 10,50 |
| 3 162 | 0,069 | 4,781 | 2,582 | 4,299 | 0,144 | 0,149 | 0,019 | 12,04 |
| 10 000 | 0,196 | 6,792 | 3,544 | 4,308 | 0,308 | 0,338 | 0,074 | 15,56 |
| 31 622 | 0,654 | 7,322 | 5,401 | 4,486 | 0,667 | 0,945 | 0,236 | 19,71 |
| 100 000 | 3,174 | 7,586 | 12,886 | 5,642 | 2,175 | 2,810 | 0,823 | 35,10 |
| 316 227 | 14,990 | 9,709 | 39,022 | 8,849 | 9,994 | 10,039 | 5,338 | 97,94 |
| 1 000 000 | 50,302 | 13,295 | 120,072 | 18,473 | 22,845 | 27,907 | 11,862 | 264,76 |

Tab. 7.6: Doba trvání triangulace na 2 GPU.

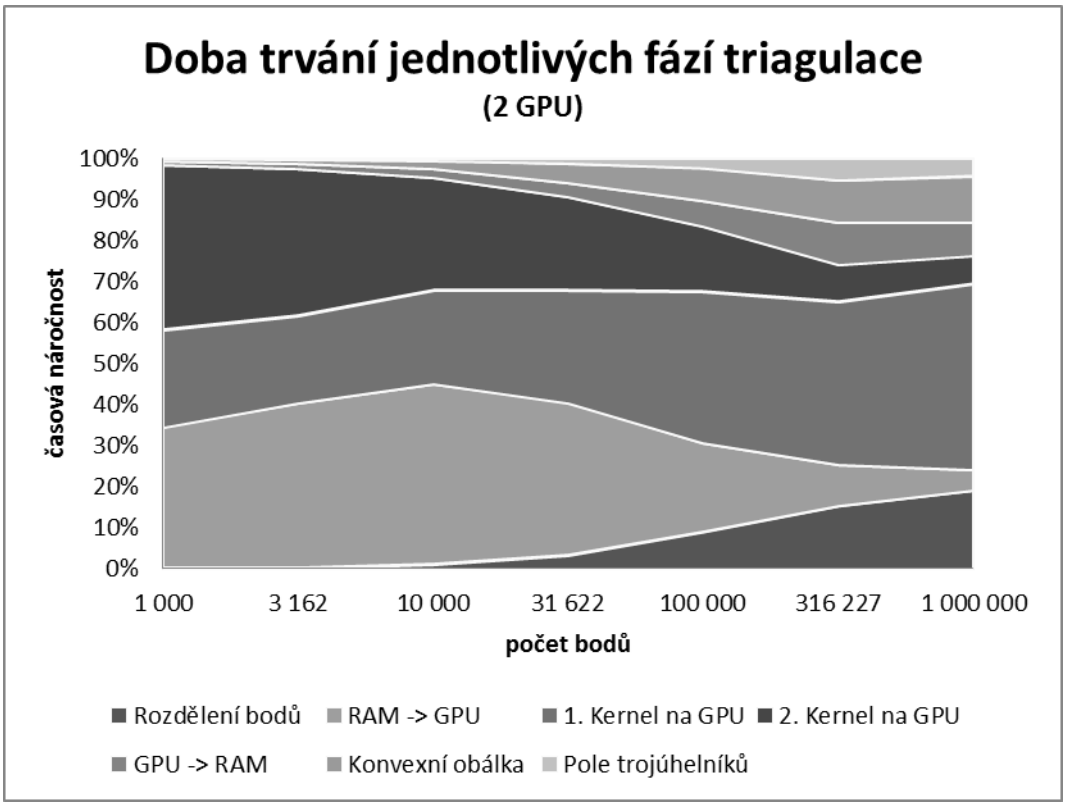
Z tabulky (Tab. 7.6) je zřejmé, že časová náročnost vykonávání jednotlivých kernelů na GPU se snížila na polovinu. Tento výsledek je shodný s tím, že vykonávané množství práce se rozdělilo na polovinu pro každé GPU. Ostatní časy, které nesouvisí s GPU jsou shodné, neboť kód, který je vykonáván pomocí CPU je shodný, jak pro jedno GPU, tak i pro dvě GPU.

V následující tabulce (Tab. 7.7) byla opět časová náročnost jednotlivých fází triangulace přepočtena na procentní podíl vzhledem k celkové době triangulace.

| | Časová náročnost [%] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Počet bodů | Rozdělení bodů | RAM -> GPU | 1. Kernel na GPU | 2. Kernel na GPU | GPU -> RAM | Konvexní obálka | Pole trojúhelníků | Celkem |
| 1 000 | 0,2% | 34,3% | 23,8% | 40,1% | 0,9% | 0,6% | 0,0% | 100% |
| 3 162 | 0,6% | 39,7% | 21,4% | 35,7% | 1,2% | 1,2% | 0,2% | 100% |
| 10 000 | 1,3% | 43,6% | 22,8% | 27,7% | 2,0% | 2,2% | 0,5% | 100% |
| 31 622 | 3,3% | 37,1% | 27,4% | 22,8% | 3,4% | 4,8% | 1,2% | 100% |
| 100 000 | 9,0% | 21,6% | 36,7% | 16,1% | 6,2% | 8,0% | 2,3% | 100% |
| 316 227 | 15,3% | 9,9% | 39,8% | 9,0% | 10,2% | 10,3% | 5,5% | 100% |
| 1 000 000 | 19,0% | 5,0% | 45,4% | 7,0% | 8,6% | 10,5% | 4,5% | 100% |

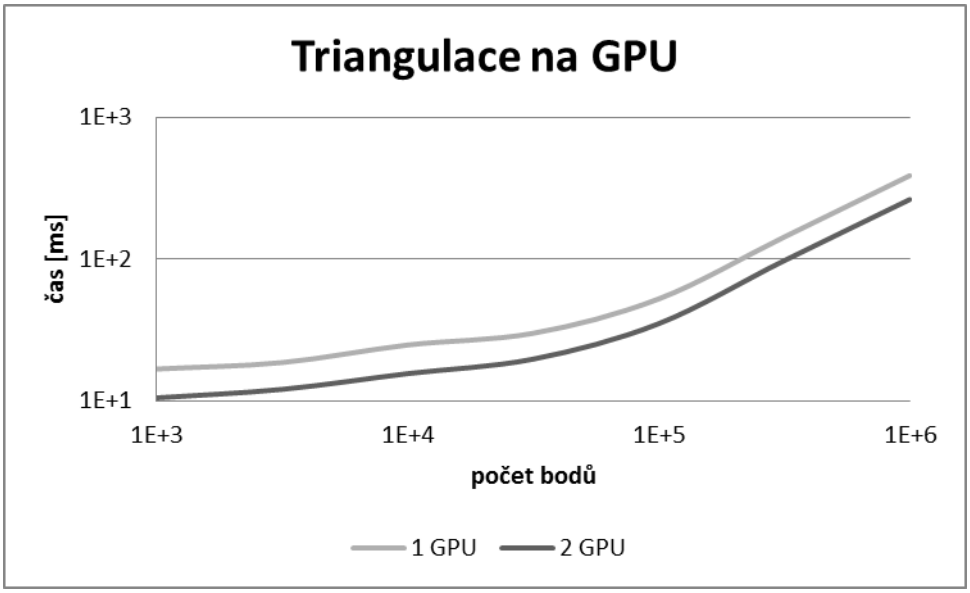
Tab. 7.7: Časová náročnost jednotlivých fází triangulace na 2 GPU.

Pomocí tabulky (Tab. 7.7) byl vytvořen graf (Graf 7.7) ve kterém jsou zobrazeny procentuální náročnosti jednotlivých fází triangulace.

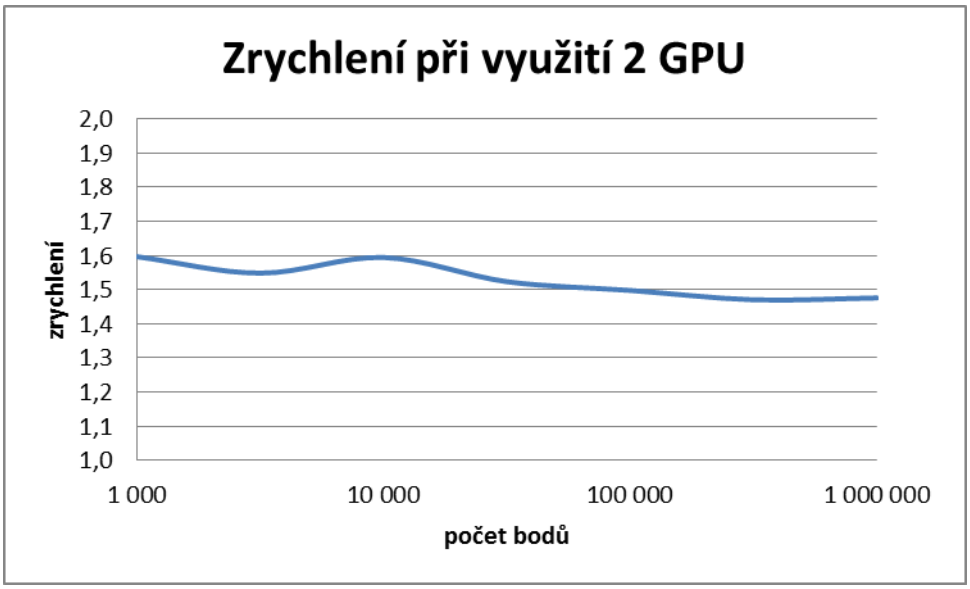



Graf 7.7: Časová náročnost jednotlivých fází triangulace na 2 GPU.

Výsledné časové náročnosti triangulací pomocí jednoho a dvou GPU je možné porovnat pomocí grafu (Graf 7.8). Je názorně vidět, že triangulace pomocí většího počtu GPU je rychlejší a poměrné zrychlení triangulace při využití dvou GPU oproti jednomu GPU je zobrazeno v grafu (Graf 7.9).



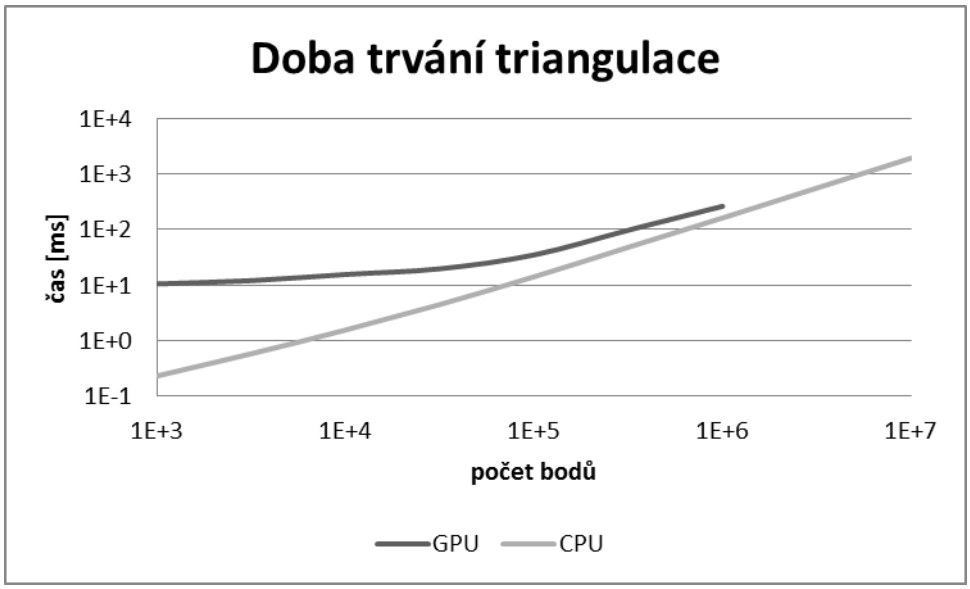

Graf 7.8: Doba trvání triangulace.




Graf 7.9: Poměrné zrychlení triangulace při využití 2 GPU oproti jednomu GPU.

#### 7.1.3.3 Porovnání CPU vs. GPU

Metoda paralelní triangulace je napsána tak, že téměř nepozměněný kód běžící pomocí CUDA na GPU lze přeložit a spustit na CPU. Díky této vlastnosti je možné porovnat jaký je rozdíl v časové náročnosti těch samých triangulací při využití GPU a CPU. Z grafu (Graf 7.10) je možné pozorovat mnohem vyšší časovou náročnost triangulace na GPU pro menší počty bodů než na CPU. To je způsobeno overheadem používání GPU (alokace a kopírování paměti, spouštění kernelu). Při větším počtu bodů se k sobě časy dvou triangulací postupně přibližují, ovšem stále je triangulace na CPU rychlejší.




Graf 7.10: Doba trvání triangulace na GPU a CPU.

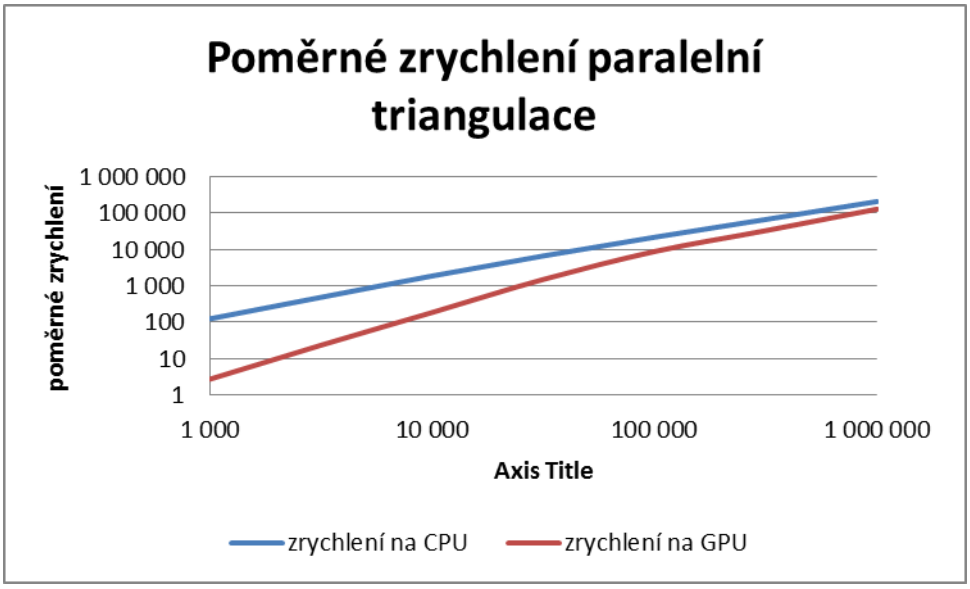
#### 7.1.3.4 Porovnání s ostatními metodami

Implementace navrhnuté paralelní triangulace využívá pro triangulaci každého gridu Delaunayovu triangulaci s očekávanou složitostí $O(N^2)$. Je vhodné porovnat, jakého urychlení bylo dosáhnuto pomocí navrhnuté paralelní metody.

$$\lambda = \frac{t_{referenční\ algoritmus}}{t_{testovaný\ algoritmus}}, \tag{7.1}$$

kde $\lambda$ je poměrné zrychlení testovaného algoritmu vůči referenčnímu algoritmu a $t_x$ je doba výpočtu algoritmu.

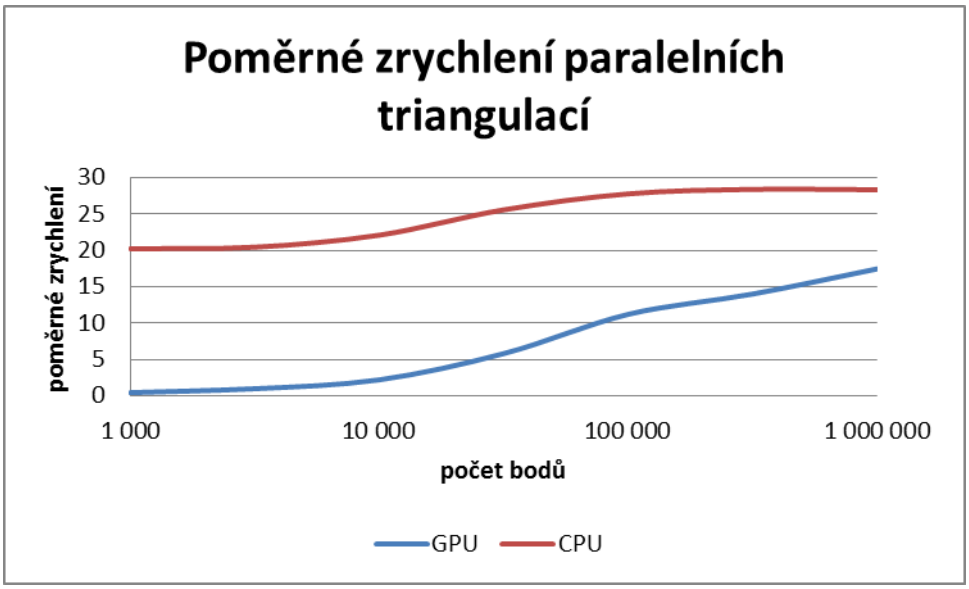
V následujícím grafu je zobrazeno zrychlení paralelních metod na CPU a GPU oproti Delaunayově triangulaci využívané pro triangulaci jednotlivých gridů.




Graf 7.11: Poměrné zrychlení paralelních triangulací oproti využívané Delaunaově triangulaci pro triangulaci jednoho gridu.

Z grafu (Graf 7.11) je názorně vidět, že při rozdělení vstupní množiny bodů na několik nezávislých množin, a poté jejich triangulace pomocí jedné metody je možné získat urychlení oproti triangulaci tou samou metodou až $10^5$ při počtu $10^6$ bodů.

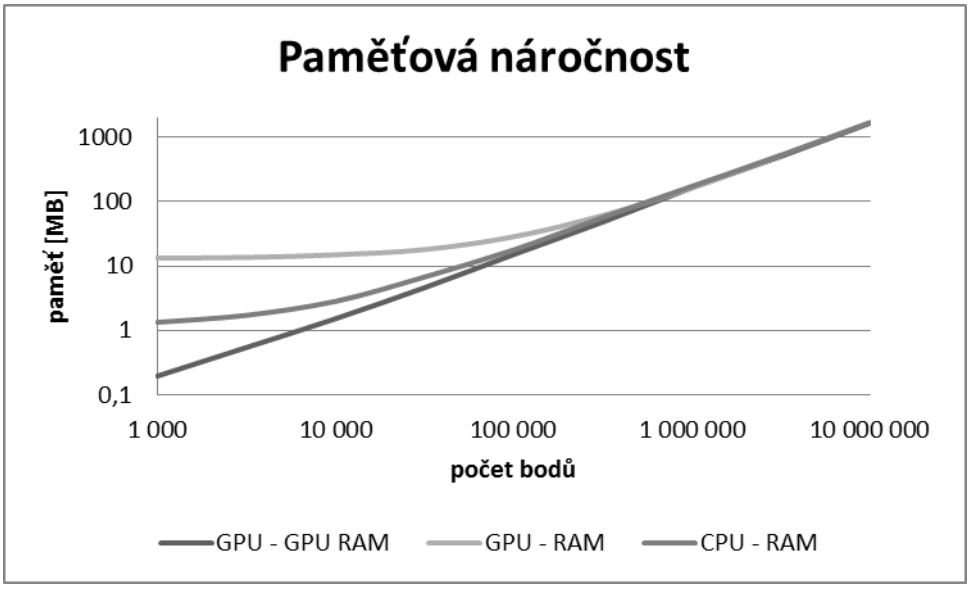
Jednou z popsaných metod paralelní triangulace je metoda z (kap. 2.1.4.2). Pro tuto metodu byly naměřeny časové náročnosti triangulace pro různě velké vstupní množiny bodů s náhodným rozložením. Výpočet triangulace probíhal pomocí 8 vláken. Po získání naměřených časových složitostí byl vytvořen graf (Graf 7.12) porovnávající poměrné zrychlení navrhnuté paralelní metody na CPU a GPU oproti této metodě z (kap. 2.1.4.2).




Graf 7.12: Poměrné zrychlení paralelních triangulací oproti paralelní metodě triangulace popsané v (kap. 2.1.4.2).

### 7.1.4 Paměťová náročnost

Jedním z dalších údajů mimo časové náročnosti, který je důležitý, je paměťová náročnost triangulace. Paměťová náročnost byla naměřena pro obě implementace paralelní triangulace. Pro triangulaci na CPU byla měřena velikost paměti RAM potřebná k běhu a pro triangulaci na GPU byla taktéž měřena velikost paměti RAM, ale poté i potřebná paměť na GPU. Graf zobrazující potřebnou paměť je zobrazen v následujícím grafu.



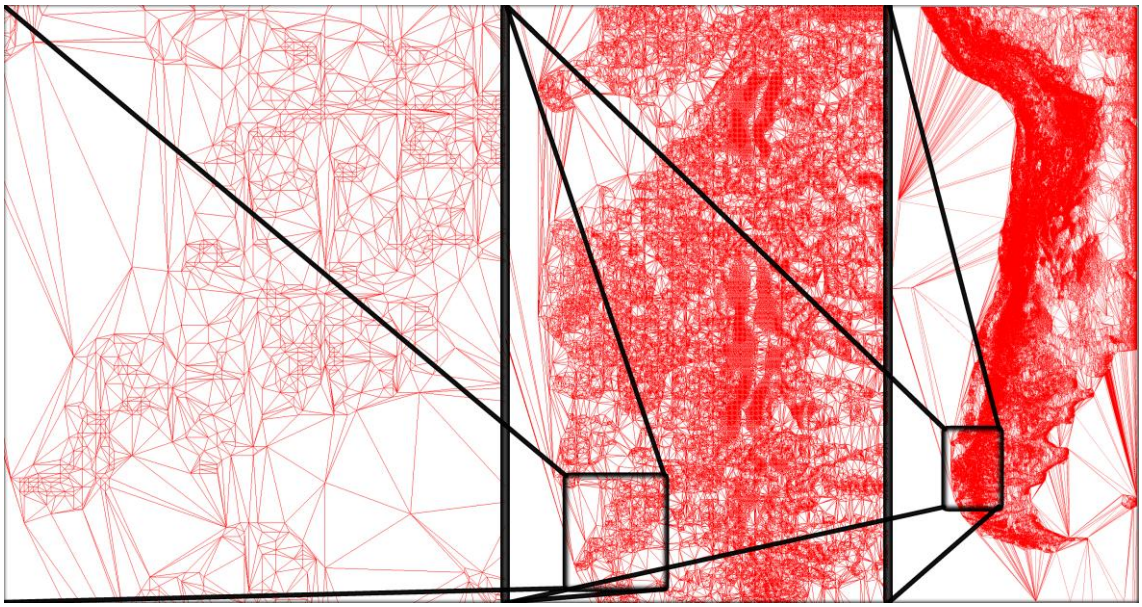

Graf 7.13: Paměťová náročnost triangulace na GPU a CPU. GPU potřebuje GPU RAM a RAM, CPU potřebuje pouze RAM.

V grafu (Graf 7.13) jsou položky $GPU\text{-}GPU\ RAM$ a $GPU\text{-}RAM$ velikosti potřebné paměti na GPU a v RAM pro triangulaci na GPU. Třetí položka $CPU\text{-}RAM$ je poté potřebná paměť pro triangulaci pomocí CPU. Je názorně vidět, že nejprve se hodnoty velikostí jednotlivých potřebných pamětí liší. S přibívajícím počtem bodů pro triangulaci se k sobě všechny tři hodnoty přibližují až od jistého počtu bodů jsou shodné. Velikost potřebné paměti (na CPU i GPU) je lineárně závislá na počtu bodů pro triangulaci.

### 7.1.5 Triangulace reálných dat

Paralelní triangulace byla testována i na reálných datech. Jako reálná data byly použity GIS data. Tyto data obsahují souřadnici bodu a jeho nadmořskou výšku, jedná se tedy o $2\frac{1}{2}D$ data.

Ukázka výsledné triangulace dat Jižní Ameriky (Chile + Argentina) je zobrazena na (Obr. 7.1). Vstupní data obsahují 1 109 932 bodů.



Obr. 7.1: Triangulace povrchového modelu Jižní Ameriky (Chile + Argentina).

## 7.2 Paralelní tetrahedronizace v $E^3$

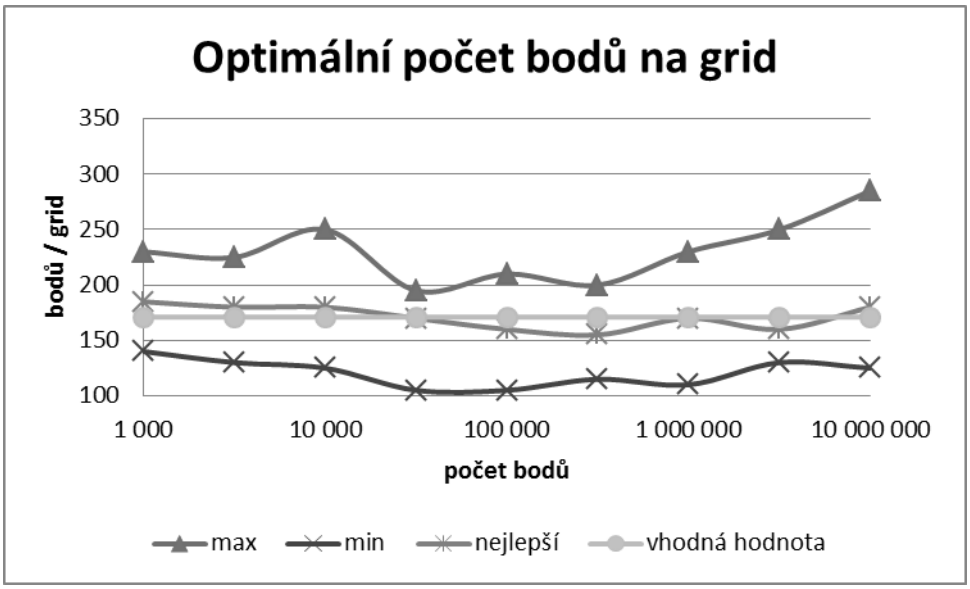
Při testování tetrahedronizace byla vstupní množina bodů vždy generována tak, že body se nacházely v prostoru $\langle -1; 1\rangle \times \langle -1; 1\rangle \times \langle -1; 1\rangle$. Body byly generovány s náhodným rozložením.

### 7.2.1 Počet bodů na grid

Při rozdělování bodů do gridu je nutné znát rozměry mřížky, která představuje rozdělení do gridů. Aby se rozměr mřížky dal vypočítat, je nutné znát průměrný počet bodů, které mají být v jednom gridu. Výsledný čas tetrahedronizace poté závisí na

zvoleném počtu bodů. Pokud by se zvolil počet bodů na grid příliš malý nebo naopak příliš velký, byla by výsledná doba tetrahedronizace zbytečně dlouhá. Úkolem je tedy najít takový počet bodů na grid, pro který bude výsledný čas tetrahedronizace minimální.

Tetrahedronizace byla otestována s různými počty bodů na grid pro různé počty vstupních bodů s náhodným rozložením. Získané výsledky byly zpracovány a vytvořeny grafy (Graf 11.22 až Graf 11.30), pro různé počty vstupních bodů, na kterých je zobrazena časová náročnost tetrahedronizace při různých počtech bodů na grid. Pomocí těchto grafů byly nalezeny intervaly počtu bodů na grid, ve kterých je doba tetrahedronizace nízká. Dalšími nalezenými hodnotami je počet bodů na grid, při kterém je doba triangulace nejnižší. Tyto získané hodnoty počtu bodů na grid jsou zobrazeny v následujícím grafu.




Graf 7.14: Souhrnné zobrazení optimálního počtu bodů na grid.

V grafu (Graf 7.14) je vidět, že nejlepší počet bodů na grid je přibližně konstantní. Jako optimální počet bodů na grid lze tedy použít hodnotu 171 bodů, což je vhodný konstantní počet bodů na grid.

### 7.2.2 Časová náročnost

Jedním z důležitých faktorů kvality tetrahedronizace je její rychlost. Testování probíhalo na množinách bodů s rovnoměrným rozložením a různým počtem. Měřeny byly postupně časové náročnosti jednotlivých částí tetrahedronizace, které jsou následující:

- Rozdělení bodů: Vstupní množina bodů je rozdělena do jednotlivých gridů.
- Tetrahedronizace gridu: Tetrahedronizace bodů příslušejících vždy jednomu gridu.

- Odebrání vnitřních bodů: Odebrání přidaných řídících bodů, které neleží na povrchu obalového kvádru všech vstupních bodů, a odebrání přidaných bodů do gridu (pokud byl prázdný).
- Konvexní obálka: Odebrání zbylých bodů řídící mřížky a vytvoření tak konvexní obálky tetrahedronizace.
- Pole tetrahedronů: Spojení jednotlivých polí tetrahedronů do výsledného pole tetrahedronů.

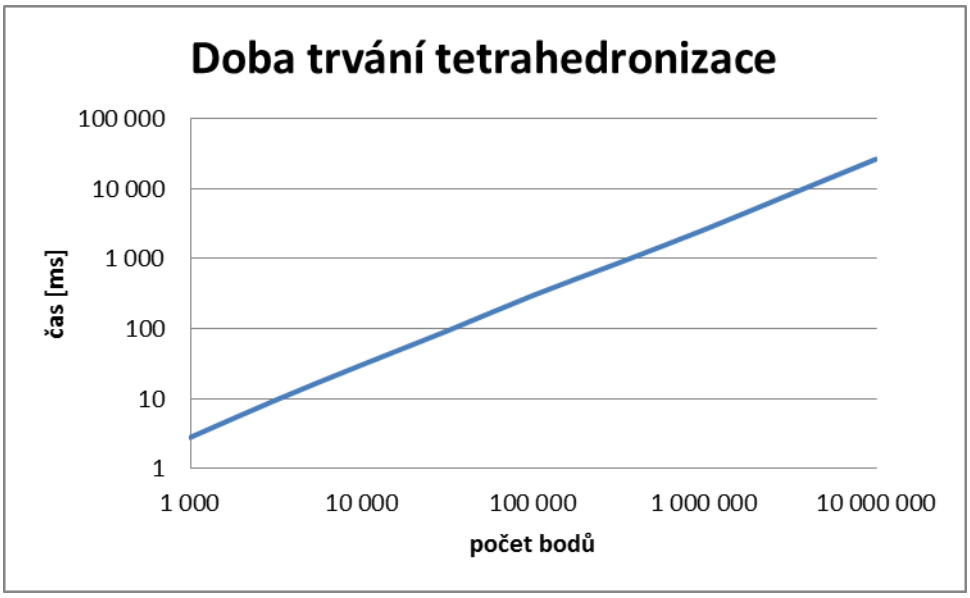
Naměřené časové náročnosti byly zpracovány a pomocí nich byla vytvořena následující tabulka.

| Počet bodů | Doba trvání [ms] | | | | | |
|---|---|---|---|---|---|---|
| | Rozdělení bodů | Tetrahedroni. gridů | Odebrání vnitřních bodů | Konvexní obálka | Pole tetrahedronů | Celkem |
| 1 000 | 0,018 | 2,377 | 0,006 | 0,378 | 0,010 | 2,8 |
| 3 162 | 0,052 | 5,453 | 0,277 | 3,601 | 0,160 | 9,5 |
| 10 000 | 0,164 | 15,066 | 2,324 | 12,023 | 0,745 | 30,3 |
| 31 622 | 0,449 | 49,282 | 6,152 | 34,599 | 2,560 | 93,0 |
| 100 000 | 1,992 | 145,094 | 23,913 | 119,131 | 9,273 | 299,4 |
| 316 227 | 7,145 | 445,673 | 72,939 | 321,891 | 24,894 | 872,5 |
| 1 000 000 | 31,198 | 1 377,690 | 186,540 | 920,592 | 95,005 | 2 611,0 |
| 3 162 277 | 115,407 | 4 324,620 | 734,558 | 2 938,090 | 233,924 | 8 346,6 |
| 10 000 000 | 465,479 | 13 703,500 | 2 039,450 | 9 509,000 | 858,456 | 26 575,9 |

Tab. 7.8: Doba trvání tetrahedronizace v závislosti na počtu vstupních bodů.

Z tabulky (Tab. 7.8) je možné pozorovat, že časová náročnost tetrahedronizace je téměř lineární. To lze zdůvodnit tak, že pokud se zvýší počet bodů pro tetrahedronizaci, pak se úměrně zvýší počet gridů i počet řídících bodů. Tím pádem se i zvýšení počtu bodů úměrně zvýší i časová náročnost celkové tetrahedronizace.

Výsledné časy trvání celkové tetrahedronizace v závislosti na velikosti množiny vstupních bodů jsou zobrazeny v následujícím grafu.




Graf 7.15: Celková doba tetrahedronizace.

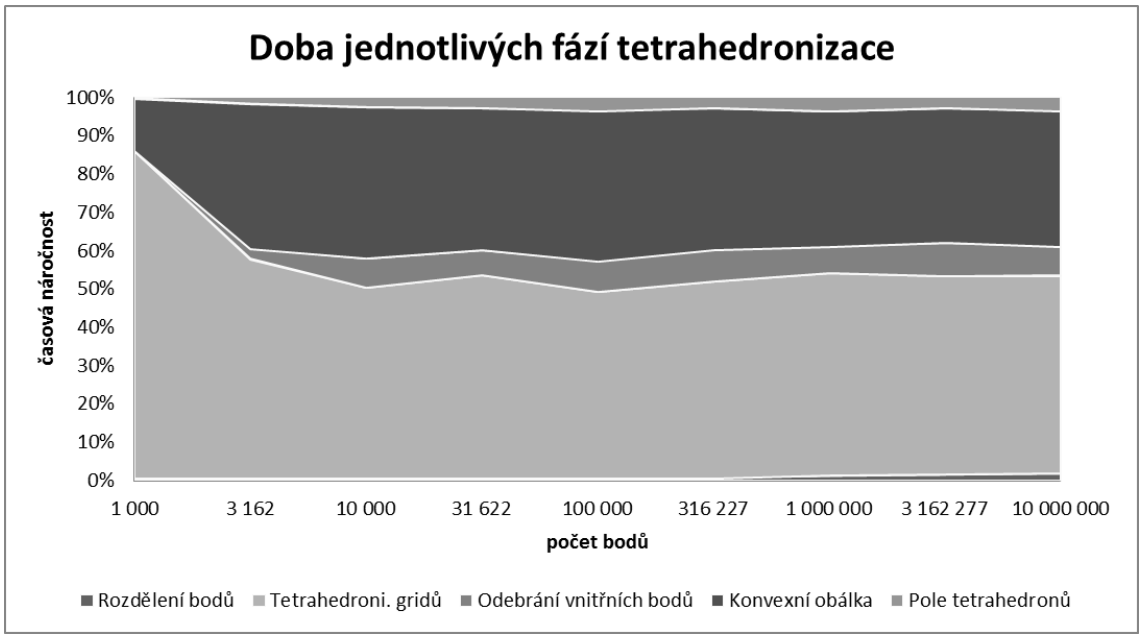
S využitím hodnot z tabulky (Tab. 7.8) lze vypočítat procentuální náročnost jednotlivých částí tetrahedronizace pro různě velké množiny bodů. V následující tabulce jsou takto vypočtené hodnoty zobrazeny.

| Počet bodů | Časová náročnost [%] | | | | | |
|---|---|---|---|---|---|---|
| | Rozdělení bodů | Tetrahedroni. gridů | Odebrání vnitřních bodů | Konvexní obálka | Pole tetrahedronů | Celkem |
| 1 000 | 0,6% | 85,2% | 0,2% | 13,6% | 0,3% | 100% |
| 3 162 | 0,5% | 57,1% | 2,9% | 37,7% | 1,7% | 100% |
| 10 000 | 0,5% | 49,7% | 7,7% | 39,7% | 2,5% | 100% |
| 31 622 | 0,5% | 53,0% | 6,6% | 37,2% | 2,8% | 100% |
| 100 000 | 0,7% | 48,5% | 8,0% | 39,8% | 3,1% | 100% |
| 316 227 | 0,8% | 51,1% | 8,4% | 36,9% | 2,9% | 100% |
| 1 000 000 | 1,2% | 52,8% | 7,1% | 35,3% | 3,6% | 100% |
| 3 162 277 | 1,4% | 51,8% | 8,8% | 35,2% | 2,8% | 100% |
| 10 000 000 | 1,8% | 51,6% | 7,7% | 35,8% | 3,2% | 100% |

Tab. 7.9: Procentuální náročnost jednotlivých částí tetrahedronizace.

Pomocí tabulky (Tab. 7.9) je možné zjistit, že polovinu z celkového času tetrahedronizace zabere tetrahedronizace bodů v gridech, která navíc běží paralelně za využití 8 vláken. Druhou nejnáročnější činností je vytvoření konvexní obálky, což zabere přibližně třetinu z celkového času tetrahedronizace. Tato činnost ovšem neběží paralelně, tudíž by při sériové verzi celé tetrahedronizace zabírala přibližně 7% celkového času.

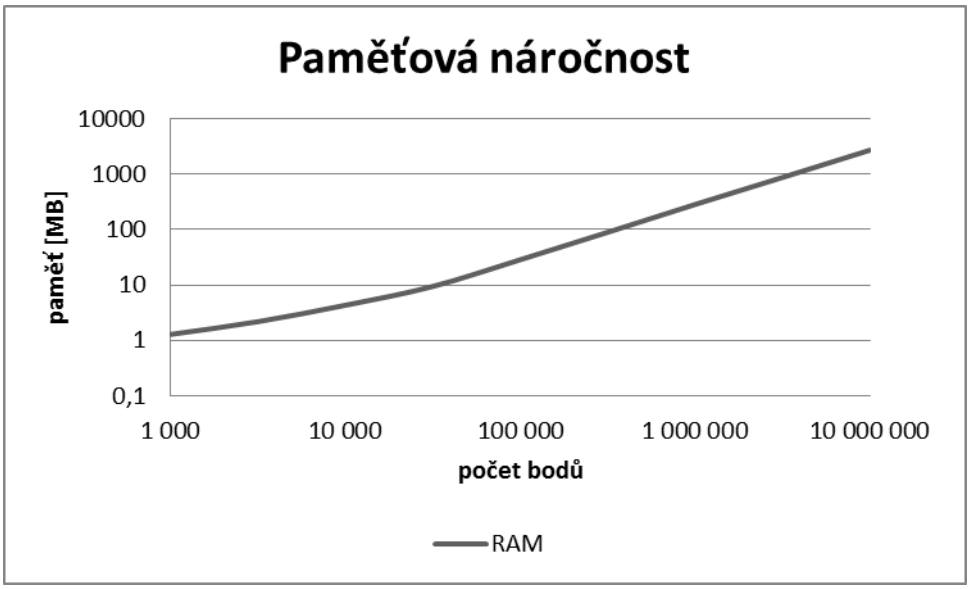
Hodnoty v tabulce (Tab. 7.9) je možné zobrazit do grafu (Graf 7.16), kde je poté názorně vidět časová složitost jednotlivých částí tetrahedronizace v závislosti na celkovém času.




Graf 7.16: Procentuální náročnost jednotlivých částí tetrahedronizace.

### 7.2.3 Paměťová náročnost

Paralelní tetrahedronizace potřebuje při svém výpočtu určité množství paměti. Program pro tetrahedronizaci byl postupně spuštěn pro různé počty vstupních bodů. Nejnižší počet bodů byl $10^3$ a poté $\sqrt{10}$ násobky až do počtu $10^7$ bodů. Pro každý počet bodů byla změřena požadovaná paměť pro běh a získané výsledky byly zpracovány do následujícího grafu.




Graf 7.17: Paměťová náročnost tetrahedronizace.

Z grafu (Graf 7.17) je možné pozorovat, že závislost velikosti potřebné paměti pro tetrahedronizaci je lineárně úměrná počtu tetrahedronizovaných bodů. Paměťová náročnost paralelní tetrahedronizace je tedy $O(N)$.

# 8 Závěr

V rámci této práce byly popsány sériové i paralelní metody triangulace a tetrahedronizace. Hlavní částí práce bylo navrhnutí nové paralelní metody triangulace bodů v $E^2$ a v $E^3$.

Mezi vstupní body pro triangulaci se vloží body definující pravidelnou mřížku. Vstupní množina bodů se poté rozdělí do jednotlivých gridů definovaných řídícími body mřížky. V dalším kroku se paralelně provedou triangulace všech gridů, neboť jednotlivé triangulace jsou na sobě nezávislé. Pokud je možné ve výsledné triangulaci ponechat navíc přidané body stačí pouze provést swap hrany mezi všemi gridy a je vytvořena výsledná triangulace. Pro dopočtení hodnot v ponechaných bodech je možné využít RBF. V opačném případě je navíc nutné přidané body z triangulace odebrat. Odebírání bodů lze opět provádět paralelně, neboť lze zajistit nezávislost při odebírání bodů.

Algoritmus triangulace bodů v $E^2$ byl nejprve implementován pomocí programovacího jazyka C#, aby se ověřila správnost navržené metody. V dalším kroku byla paralelní triangulace implementována pro výpočet na GPU s využitím CUDA a C++. Výpočet triangulace je možný rozdělit mezi více GPU, k čemuž bylo využito OpenMP. Implementovaná triangulace pro GPU není schopná triangulovat vstupní body s nerovnoměrným rozložením, neboť není možné dynamicky alokovat paměť na GPU za běhu kernelu. Pomocí podmíněného překladu je možné implementaci triangulace pro GPU přeložit také pouze pro CPU. Algoritmus tetrahedronizace bodů v $E^3$ byl implementován v C++ s využitím OpenMP pro paralelizaci. Implementace umožňuje tetrahedronizovat body s libovolným rozložením.

Pro implementované paralelní triangulace a tetrahedronizace byl pomocí testů určen optimální počet bodů v jednom gridu. Pro testovací sestavu počítače byl nalezený počet bodů vždy konstantní a nezávislý na počtu vstupních bodů pro triangulaci (tetrahedronizaci). Přesná hodnota optimálního počtu bodů ovšem závisí na dané sestavě PC.

Hlavním důvodem paralelizace je vždy snaha o snížení časové náročnosti algoritmu. Jedním z provedených testů bylo zjištění časové náročnosti triangulací a tetrahedronizace pro různé počty vstupních bodů. Měření byla prováděna pro počet vstupních bodů v rozmezí od $10^3$ až do $10^6$ ($10^7$). Z naměřených výsledků vyplynulo, že časová náročnost navržených metod je $O(N)$. Při $k$ zvětšení počtu vstupních bodů se $k$ krát zvětší počet gridů, tím pádem triagulace gridů bude trvat $k$ krát déle. To samé platí pro počet spojování gridů. Poslední částí navržené metody je vždy tvorba konvexní obálky, jejíž časová náročnost se také $k$ krát zvětší, neboť se $k$ krát zvětšil počet řídících bodů využívaných pro tvorbu konvexní obálky.

Pomocí přidání dodatečných řídících bodů mezi množinu vstupních bodů se docílilo jejího jednoduchého rozdělení a po provedení triangulací (tetrahedronizací), na množinách bodů, jejich jednoduchého spojení ve výslednou triangulaci (tetrahedronizaci).

Tato práce sloužila k ověření navrhnuté paralelní triangulace (tetrahedronizace) a z tohoto důvodu byl pro triaungulaci (tetrahedronizaci) použit algoritmus brutální síly. V dalším postupu bude tento algoritmus zaměněn za jiný efektivní algoritmus triangulace (tetrahedronizace).

V navazující práci je možné upravit pravidelné dělení vstupních bodů na dynamické dělení. Pokud bude v gridu stále příliš mnoho bodů je možné jej dále rozdělit na další dvě nebo čtyři části. Díky této úpravě by časová náročnost triangulace (tetrahedronizace) nebyla závislá na rozložení vstupních bodů.

# 9 Přehled zkratek a pojmů

CPU: procesor ( = Central Processing Unit)

GPU: grafický procesor ( = Graphic Processing Unit)

$E^2$: dvourozměrný prostor, kde body mají souřadnice $[x, y]^T$

$E^3$: trojrozměrný prostor, kde body mají souřadnice $[x, y, z]^T$

DT(P): Delaunayova triangulace množiny bodů $P$

CH(P): Konvexní obálka množiny bodů $P$

RBF: Radiální bázové funkce

# 10 Literatura


[Bay] **Bayer, Tomáš.** *Rovinné triangulace a jejich využití.* Praha: Přírodovědecká fakulta UK.

[Car12] **Carrigan, Braxton.** *Disertační práce – Triangulations and Simplex Tilings of Polyhedra.* Alabama: Auburn University, srpen 2012.

[Cig93] **Cignoni, P., Montani, C., Perego, R. a Scopigno, R.** *Parallel 3D Delaunay Triangulation.* Computer Graphics Forum, Volume 12, Issue 3, s. 129-142, srpen 1993.

[Cig98] **Cignoni, P., Montani, C. a Scopigno, R.** *DeWall: A Fast Divide & Conquer Delaunay Triangulation Algorithm in $E^d$*. Computer-Aided Design, Volume 30, Issue 5, s. 333-341, duben 1998.

[Dai05] **Dai, Meizhong a Schmidt, David P.** *Adaptive tetrahedral meshing in free-surface flow*, Journal of Computational Physics, Volume 208, Issue 1, s. 228-252, 1. září 2005.

[Fuk06] **Fuksa, Miroslav.** *Diplomová práce – Delaunayova triangulace s omezením (CDT) v $E^2$ a $E^3$*. Plzeň: Fakulta aplikovaných věd ZČU, 2006.

[Hla02] **Hlavatý, Tomáš a Skala, Václav.** *The Brute Force Generator of Triangulations with Required Properties.* ICCVG2002 Int.Conf., Poland, s. 325-336, ISBN: 839176830-9, 2002.

[Che11] **Chen, Min-Bin.** *A parallel 3D Delaunay triangulation method.* 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications, s. 52-56, 2011.

[Koh02] **Kohout, Josef.** *Diplomová práce – Paralelní Delaunayova triangulace ve 2D a 3D.* Plzeň: Fakulta aplikovaných věd ZČU, 2002.

[KoJ05] **Kohout, Josef.** *Disertační práce – Delaunay triangulation in parallel and distributed environment.* Plzeň: Fakulta aplikovaných věd ZČU, 2005.

[Koh05] **Kohout, Josef, Kolingerová, Ivana a Žára, Jiří.** *Parallel Delaunay triangulation in $E^2$ and $E^3$ for computers with shared memory.* Parallel Computing, Volume 31, s. 491-522, ISSN 0167-8191, 2005.

[Kol] **Kolingerová, Ivana.** *Triangulace polygonu, dělení polygonu na konvexní části, teorém obrazové galerie.* Plzeň: Fakulta aplikovaných věd ZČU.

[Led05] **Ledoux, Hugo, Gold, Chritopher M. a Baciu, George.** *Flipping to Robustly Delete a Vertex in a Delaunay Tetrahedralization.* Computational Science and Its Applications – ICCSA 2005, Volume 3480, s. 737-747, 2005.

[Lei06] **Leifer, Filip.** *Diplomová práce – Delaunayova triangulace a její aplikace.* Brno: FSI ústav automatizace a informatiky VUT, 2006.

[Liu05] **Liu, Yuan Xin a Snoeyink, Jack.** *A Comparison of Five Implementations of 3D Delaunay Tessellation.* Combinatorial and Computational Geometry, MSRI Publications, Volume 52, s. 439-458, 2005.

[Mul08] **Mulzer, Wolfgang a Rote, Günter.** *Minimum-weight triangulation is NP-hard.* Journal of the ACM, Volume 55, No. 2, Article 11, květen 2008.

[NVi] **NVIDIA.** *Parallel Programming and Computing Platform CUDA,* http://www.nvidia.com/object/cuda_home_new.html [online] [citace: 7. 4. 2013].

[OMP] **OpenMP.** *The OpenMP API specification for parallel programming,* http://openmp.org [online] [citace: 6. 4. 2013].

[Sei95] **Seidel, Raimund.** *The upper bound theorem for polytopes: an easy proof of its asymptotic version.* Computational Geometry, Volume 5, Issue 2, s. 115-116, září 1995.

[Sch04] **Schaller, Gernot a Meyer-Hermann, Michael.** *Kinetic and Dynamic Delaunay tetrahedralizations in three dimensions.* Computer Physics Communications, Volume 162, Issue 1, s. 9-23, 1. září 2004.

[Ska12] **Skala, Václav.** *Metody triangulace a tetrahedronizace s vkládanými body a dělením prostoru.* Pracovní podklady. Plzeň: Fakulta aplikovaných věd ZČU, 2012.

[SkV12] **Skala, Václav.** *Radial Basis Functions for High Dimensional Visualization.* VisGra - ICONS12, Saint Gilles, Reunion Island, IARIA, s. 218-222, ISBN: 978-1-61208-184-7, 2012.

[SDX] **SlimDX.** *SlimDX Homepage.* http://slimdx.org [online] [citace: 26. 4. 2013].

[Šmo12] **Šmolík, Michal a Karlíček Lukáš.** *Semestrální práce z VID - Nová triangulační metoda založená na rozděl a panuj.* Plzeň: Fakulta aplikovaných věd ZČU, 2012.

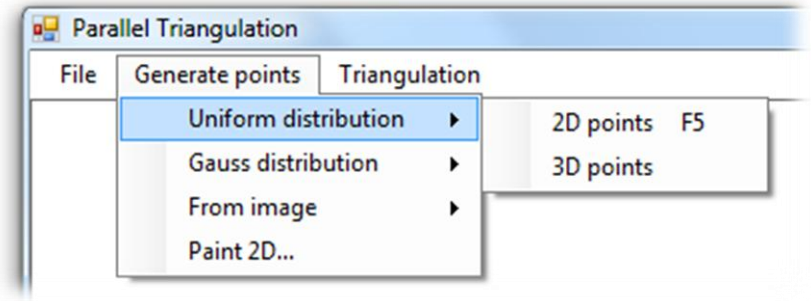
[Záb05] **Zábranský, Jaromír.** *Diplomová práce - Triangulace povrchů a úlohy na nich.* Plzeň: Fakulta aplikovaných věd ZČU, 2005.

# 11 Přílohy

# A Uživatelská příručka

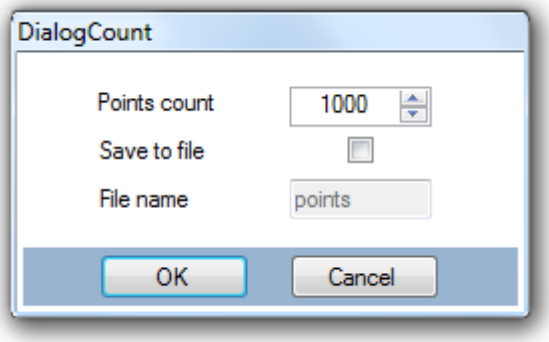
## A.1 Generování bodů

Pomocí aplikace je možné automaticky vygenerovat body a náhodným nebo gaussovským rozložením. Pro vygenerování bodů je nutné zvolit v menu položku $Uniform\ distribution$ ($Gauss\ distribution$) → $2D\ points$ (viz Obr. 11.1).




Obr. 11.1: Generování bodů.

Po kliknutí na příslušné tlačítko $2D\ points$ se zobrazí dialog, ve kterém je možné zvolit počet generovaných bodů (viz Obr. 11.2). V případě, že je požadováno vygenerované body pouze uložit do souboru a netriangulizovat, je nutné zaškrtnout políčko $Save\ to\ file$ a vyplnit jméno výstupního souboru. Vygenerovaný soubor s body bude mít koncovku $*.point2D$.

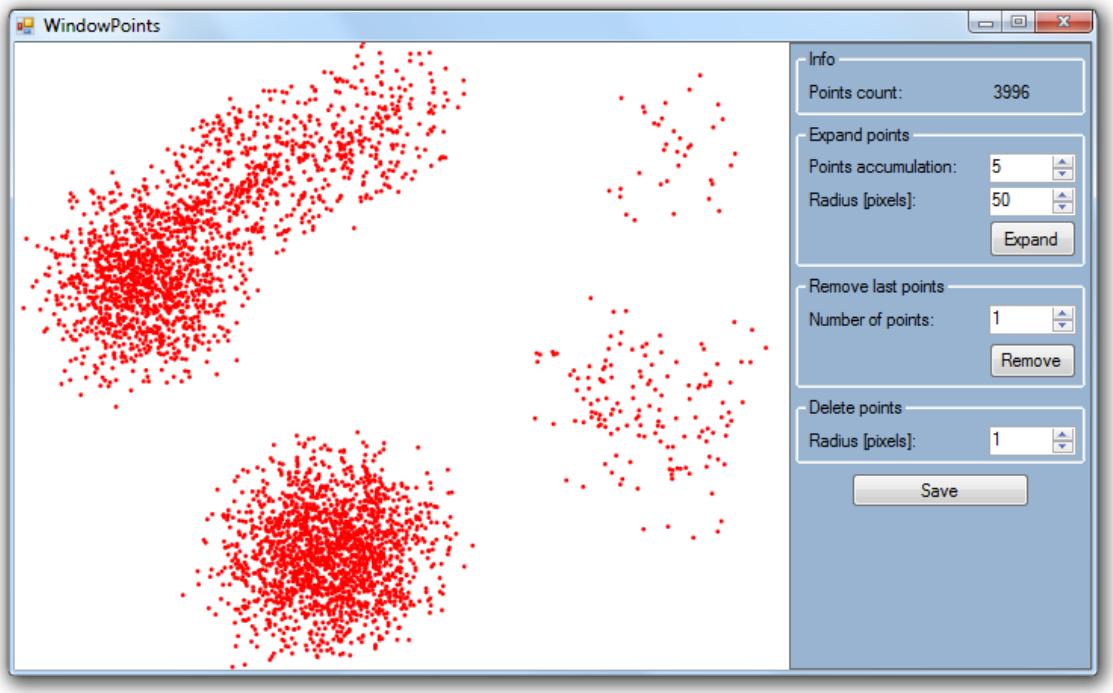



Obr. 11.2: Počet generovaných bodů.

Další možností generování bodů je s využitím vstupního obrázku. Po kliknutí na $From\ image\ \rightarrow\ 2D\ points$ je nutné zvolit vstupní obrázek v běžném formátu ($*.jpg$, $*.png$, $*.gif$ nebo $*.bmp$). Vstupní obraz je převeden na černobílý a každý pixel, jehož intenzita je větší než 50%, je převeden na 2D bod. Výstupem zpracování obrázku

je vstupní soubor s body $*.point2D$, který je umístěn ve stejné složce jako vstupní obrázek.

Poslední možností generování bodů pro triangulaci je ruční vygenerování. Po kliknutí na $Paint\ 2D$ ... se zobrazí okno pro generování bodů (viz Obr. 11.3). V levé části okna je možné klikáním přidávat body.




Obr. 11.3: Okno pro ruční generování bodů.

Body zobrazené v levé části okna je možné použít jako střed pro nově vygenerované body s gaussovským rozložením. Pro expandování bodů je nutné zvolit počet bodů, které budou kolem každého bodu vygenerovány a poloměr vzdálenosti (v pixelech). Po stisknutí pravého tlačítka je možné mazat zobrazené body.

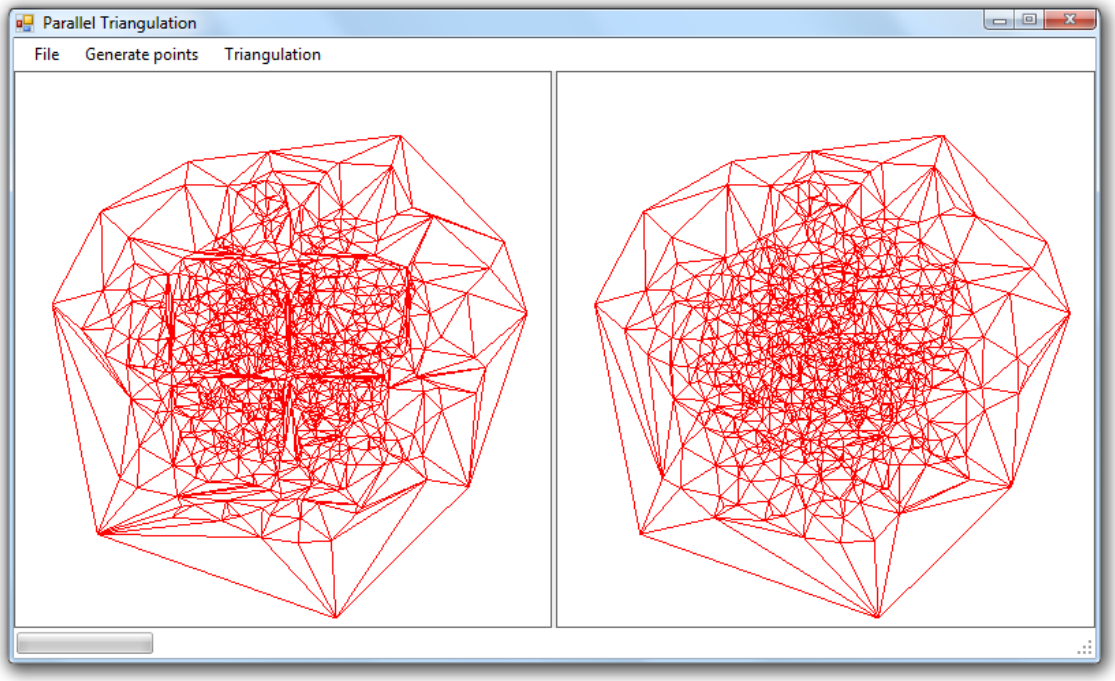
Po dokončení generování bodů je možné výsledné rozložení bodů uložit do souboru $*.point2D$ pro budoucí zpracování.

Body ze souboru $*.point2D$ nebo $*.triangle2D$ je možné načíst a ztriangulizovat. Samozřejmostí je i uložení bodů do souboru. Pro načtení (uložení) bodů je nutné zvolit v hlavním menu programu $File \rightarrow Open\ (Save)$.

## A.2 Triangulace

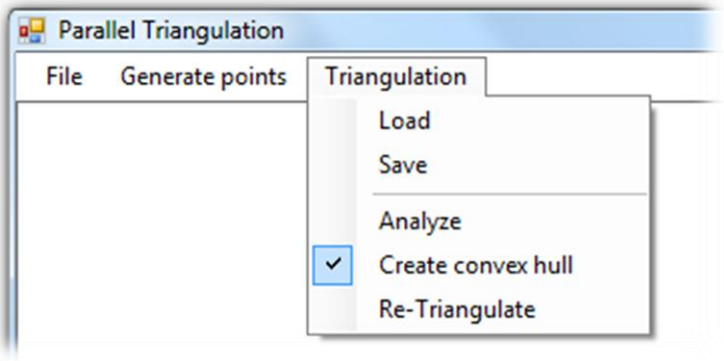
Program umožňuje porovnání triangulace vytvořené pomocí navrhnuté paralelní metody a Delaunayovy triangulace. V levé části okna (viz Obr. 11.4) je zobrazena triangulace vytvořená pomocí paralelní metody a v pravé části je zobrazena Delaunayova triangulace. Pomocí kolečka myši je možné přibližovat a oddalovat

triangulaci. Pomocí kláves $W$, $S$, $A$ a $D$ je možné s triangulací pohybovat do všech čtyř stran.




Obr. 11.4: Hlavní okno programu. Zobrazení dvou triangulací pro možnost porovnání (vlevo: paralelní metoda, vpravo: Delaunayova triangulace).

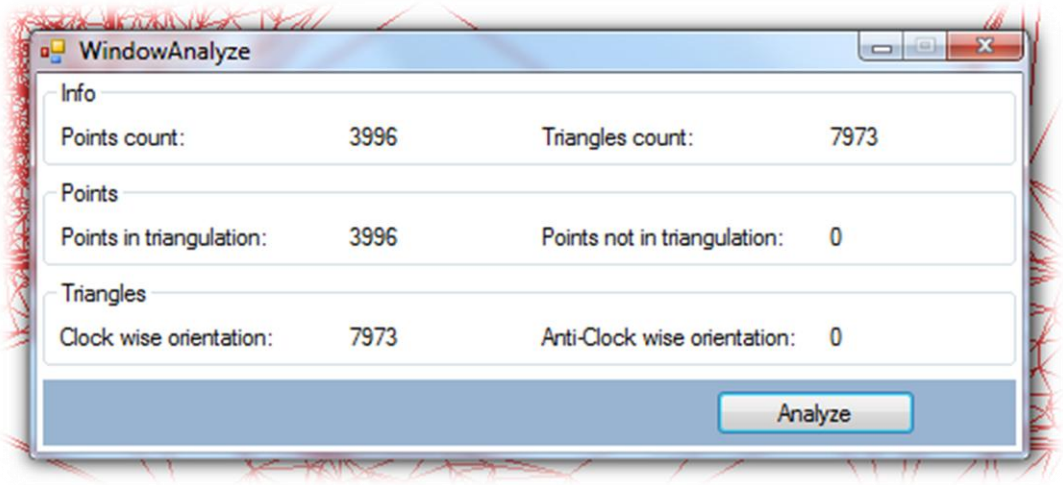
Po kliknutí volby $Triangulation$ v hlavním menu (viz Obr. 11.5) je možné provádět různé operace s triangulací. Triangulaci vytvořenou pomocí paralelní metody je možné uložit ($Save$) do souboru nebo načíst ($Load$) ze souboru, který má koncovku $*.triangle2D$.




Obr. 11.5: Menu pro práci s triangulací.

Pomocí volby $Analyze$ je možné zobrazit informace o triangulaci (viz Obr. 11.6). Mezi informace patří počet bodů a trojúhelníků, počet bodů, které jsou (a nejsou)

zahrnuty v triangulaci, a počet trojúhelníků s orientací po (proti) směru hodinových ručiček.

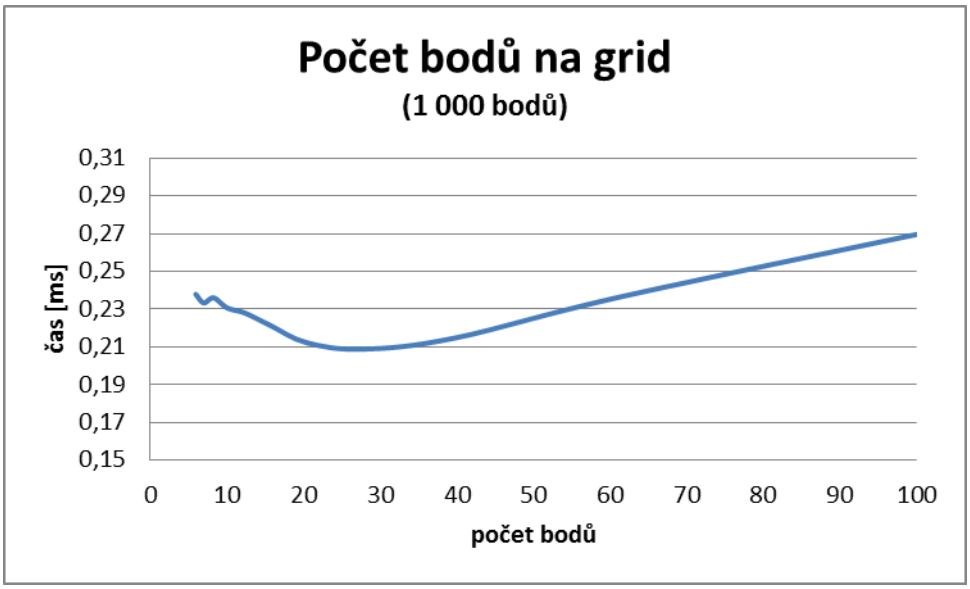



Obr. 11.6: Okno s informacemi o triangulaci.

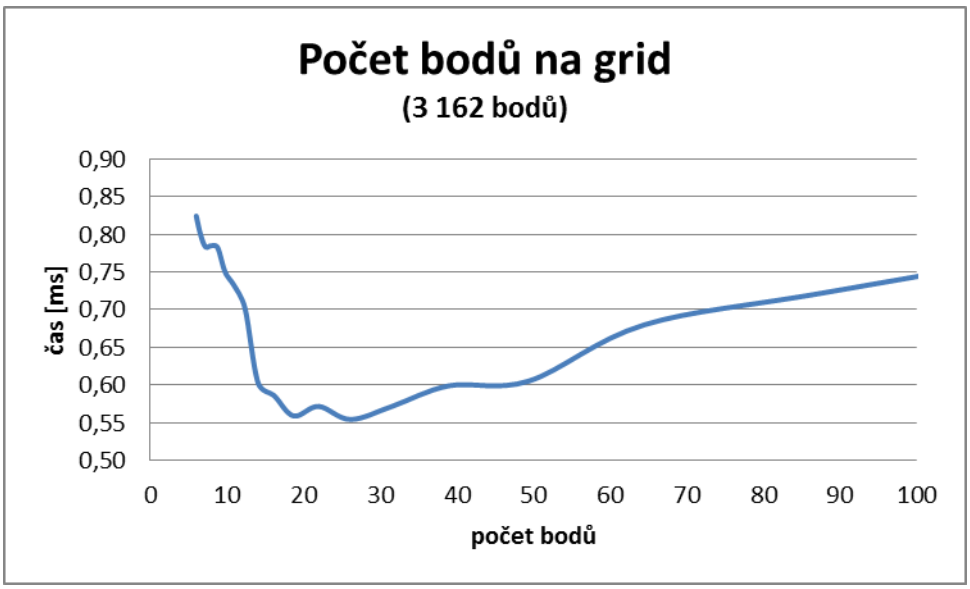
# B Grafy - triangulace

## B.1 Počet bodů na grid

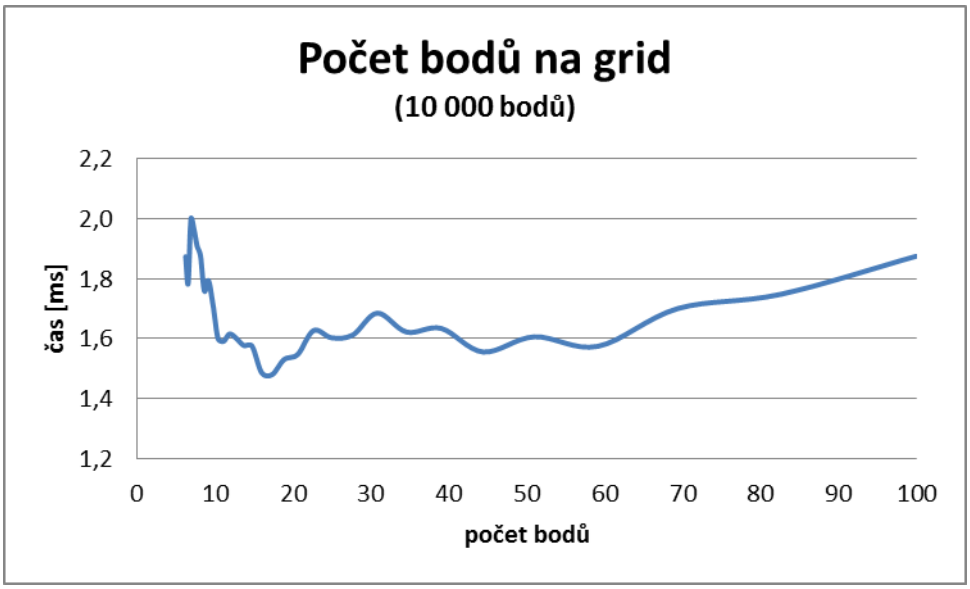
### B.1.1 Testování na CPU



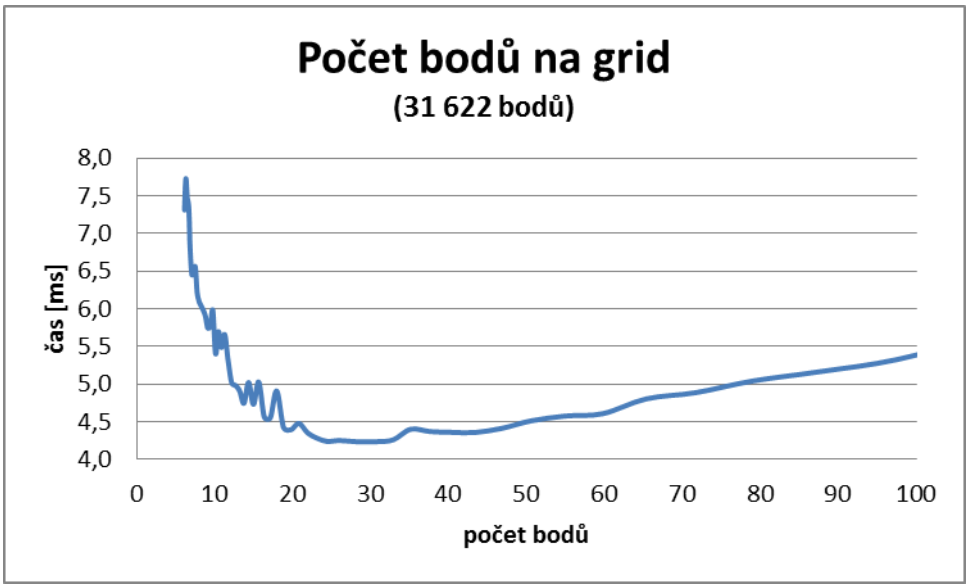

Graf 11.1: Časová náročnost v závislosti na počtu bodů na grid.



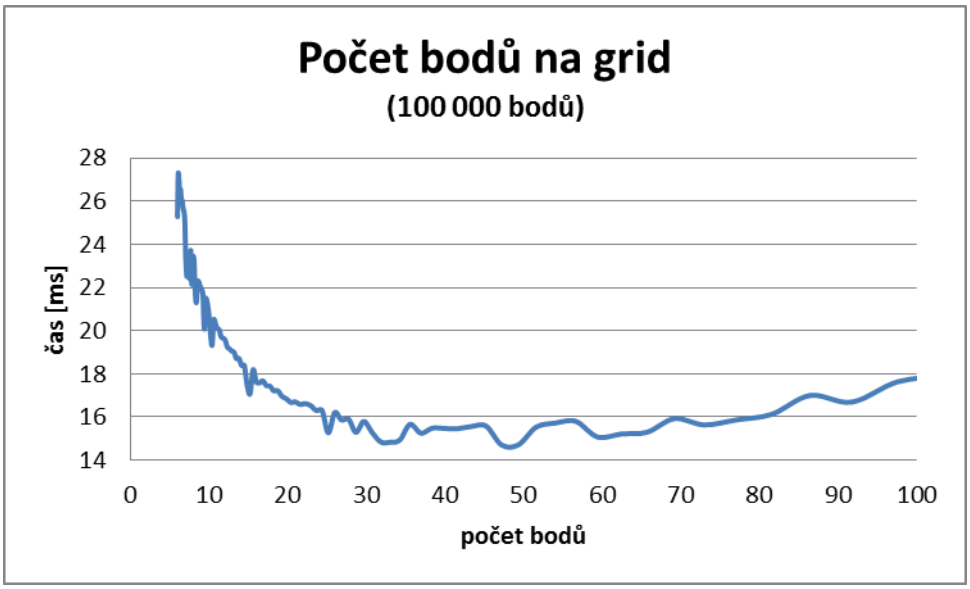

Graf 11.2: Časová náročnost v závislosti na počtu bodů na grid.



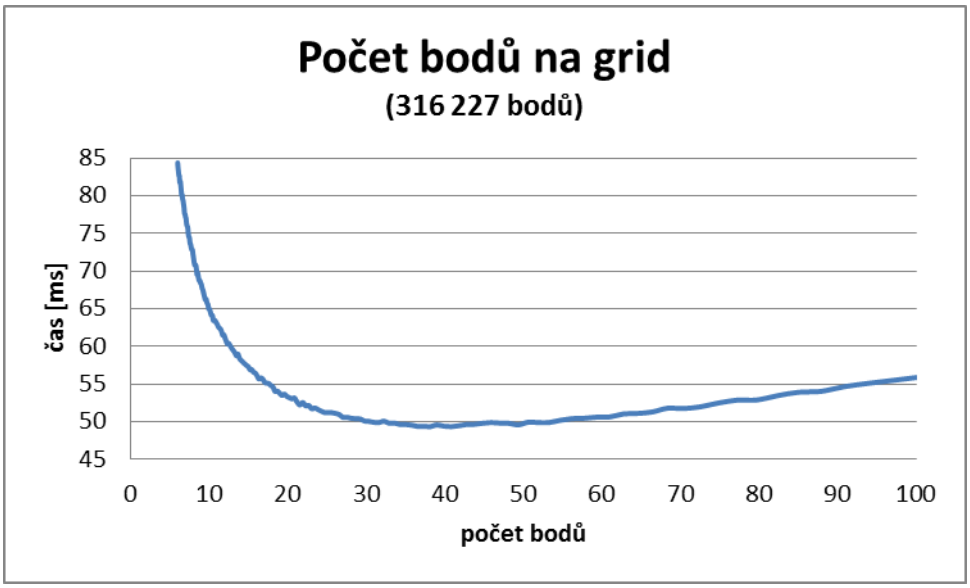

Graf 11.3: Časová náročnost v závislosti na počtu bodů na grid.



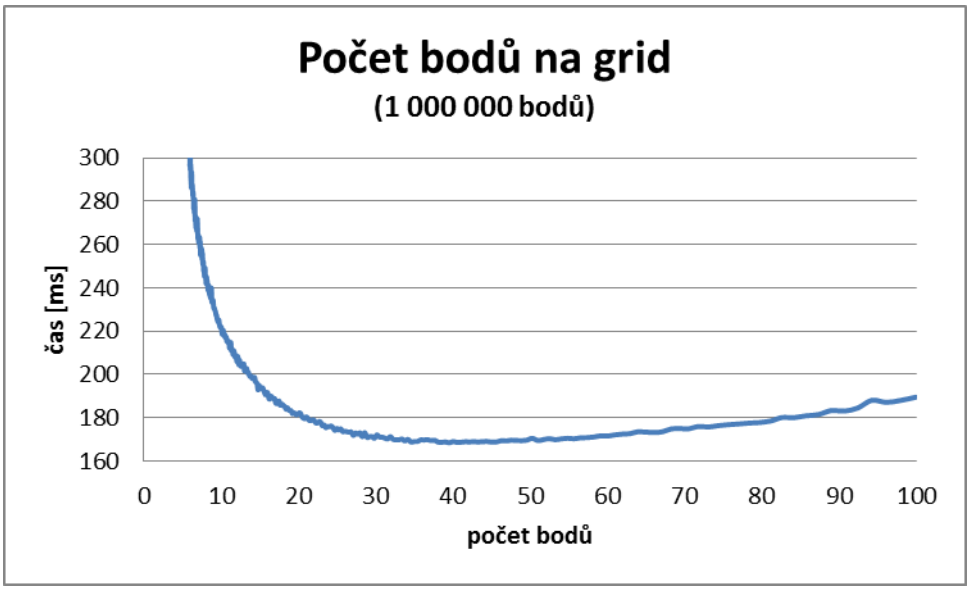

Graf 11.4: Časová náročnost v závislosti na počtu bodů na grid.




Graf 11.5: Časová náročnost v závislosti na počtu bodů na grid.

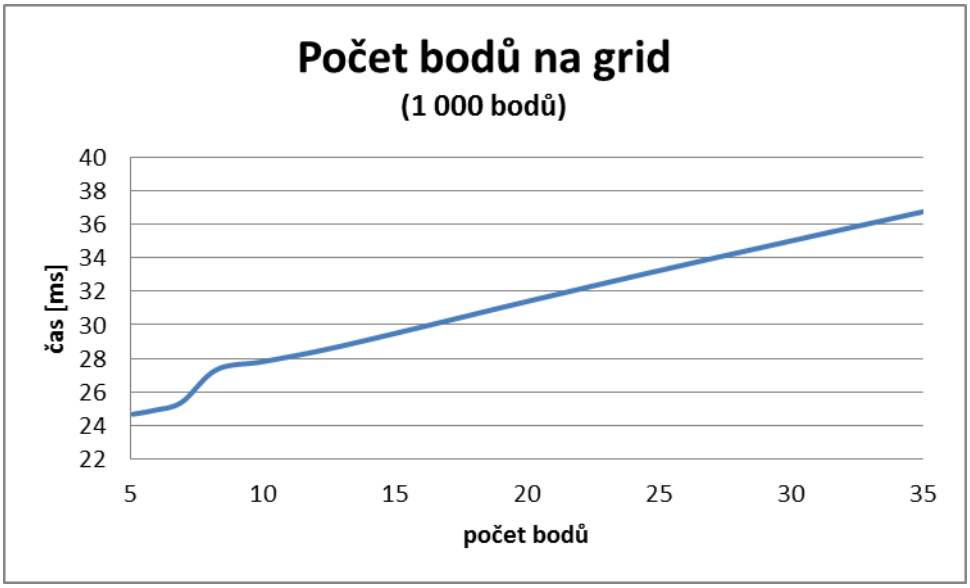



Graf 11.6: Časová náročnost v závislosti na počtu bodů na grid.

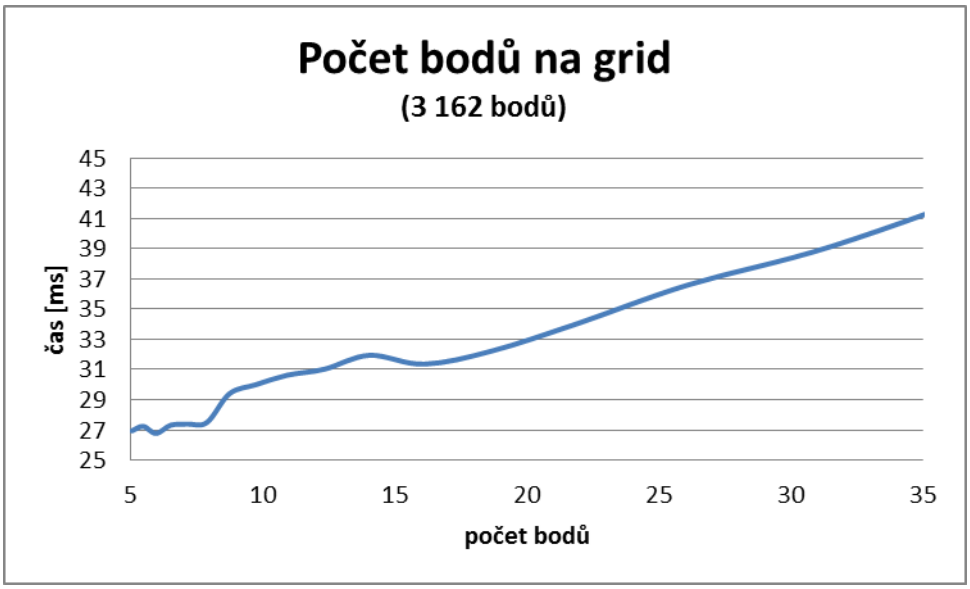



Graf 11.7: Časová náročnost v závislosti na počtu bodů na grid.

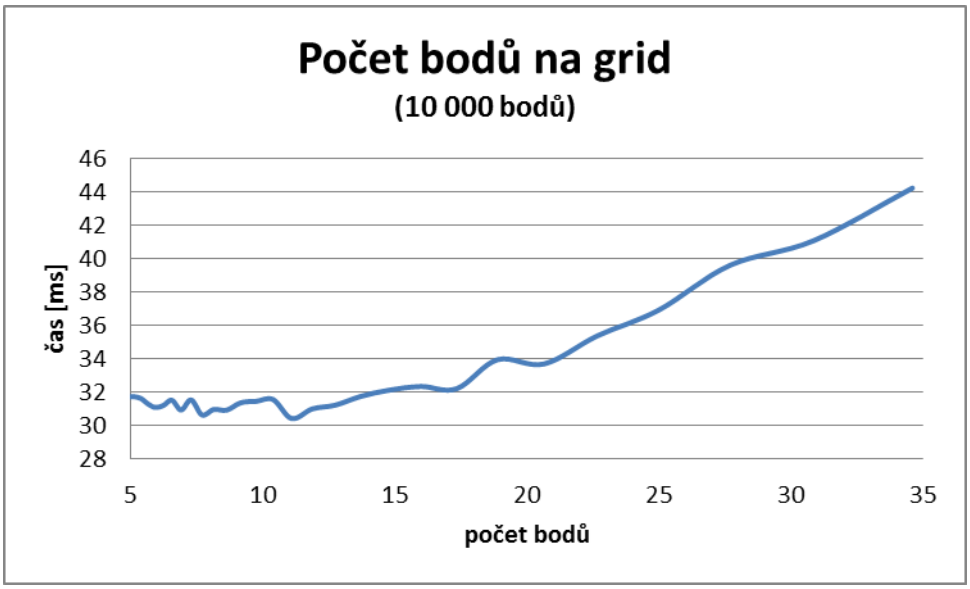
### B.1.2 Testování na GPU



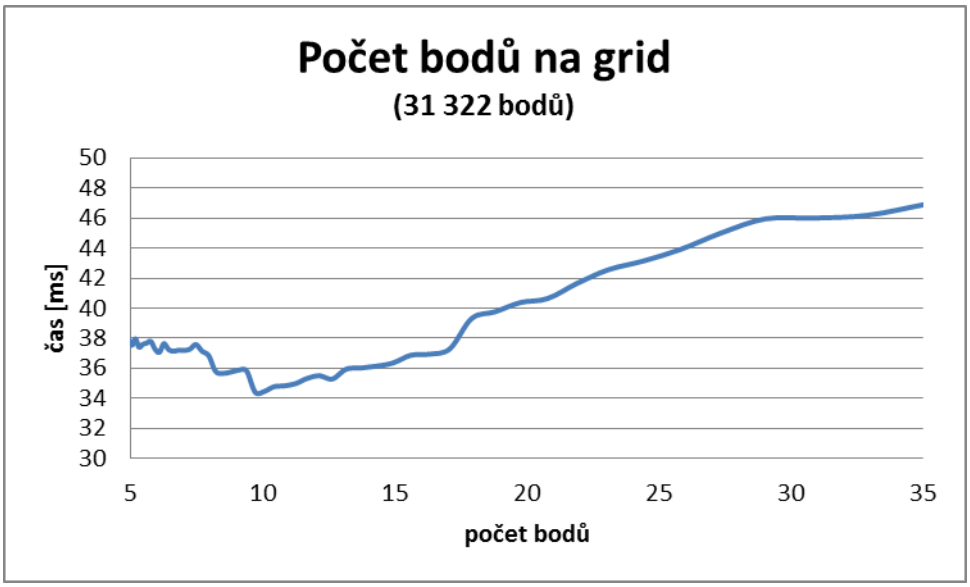

Graf 11.8: Časová náročnost v závislosti na počtu bodů na grid.



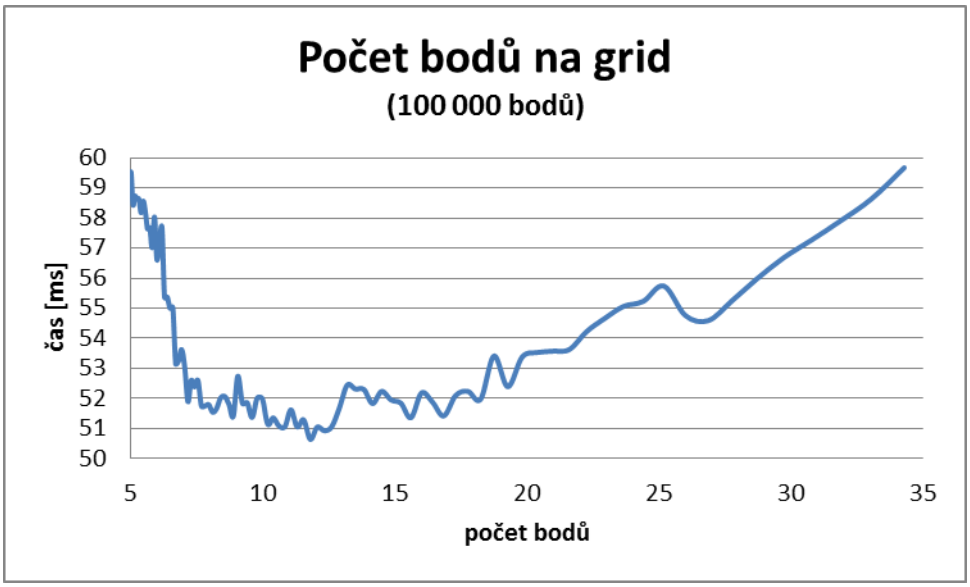

Graf 11.9: Časová náročnost v závislosti na počtu bodů na grid.



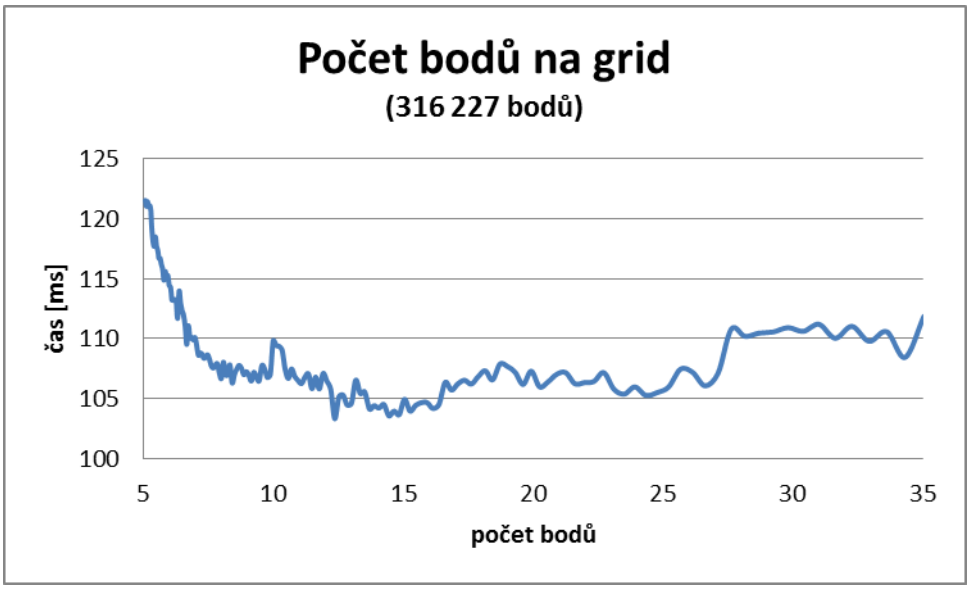

Graf 11.10: Časová náročnost v závislosti na počtu bodů na grid.



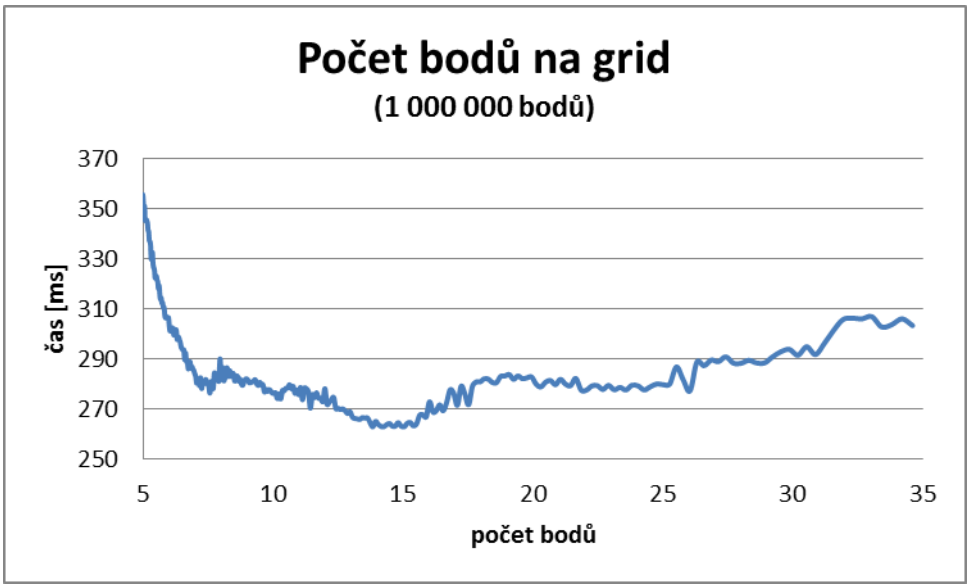

Graf 11.11: Časová náročnost v závislosti na počtu bodů na grid.




Graf 11.12: Časová náročnost v závislosti na počtu bodů na grid.

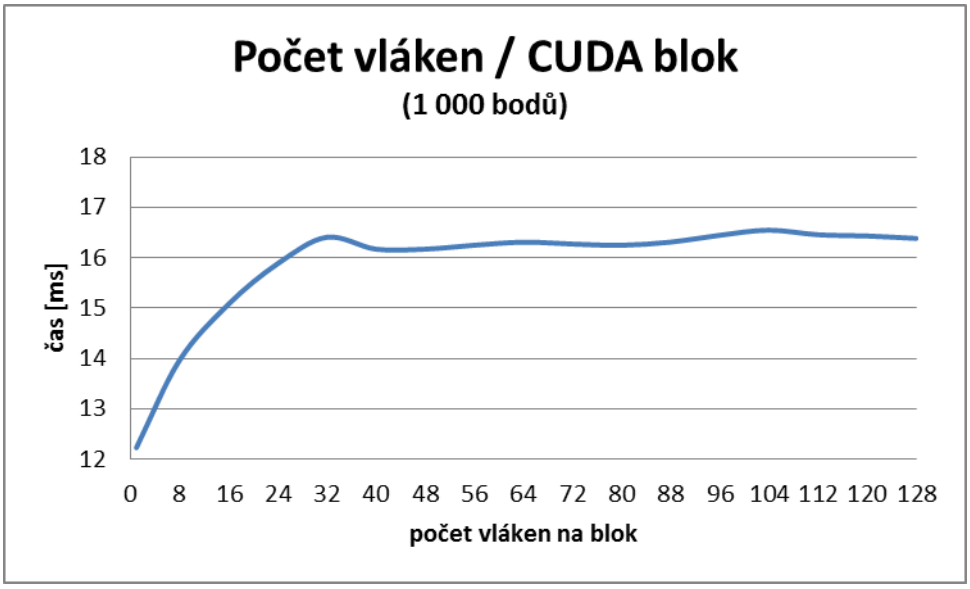



Graf 11.13: Časová náročnost v závislosti na počtu bodů na grid.

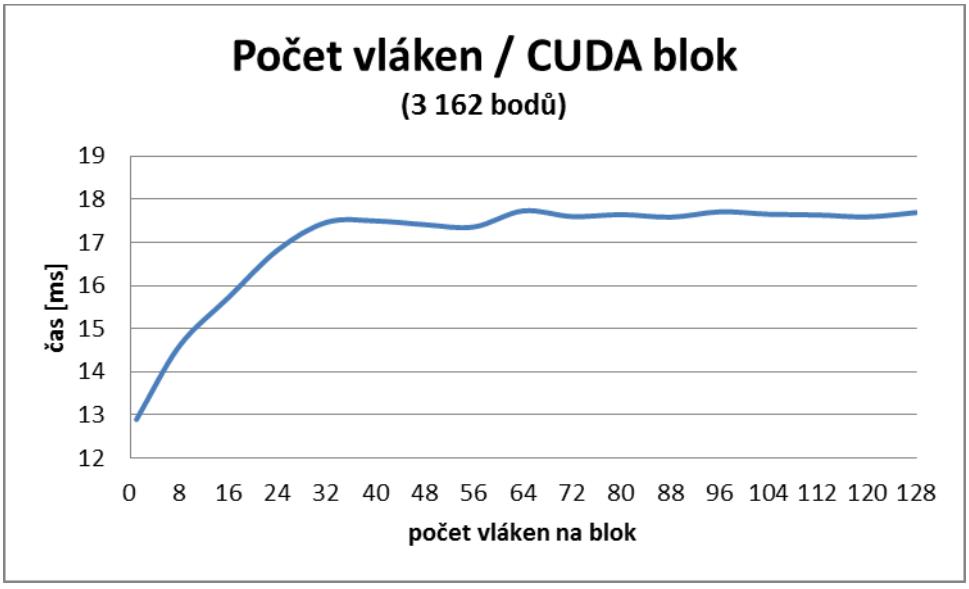



Graf 11.14: Časová náročnost v závislosti na počtu bodů na grid.

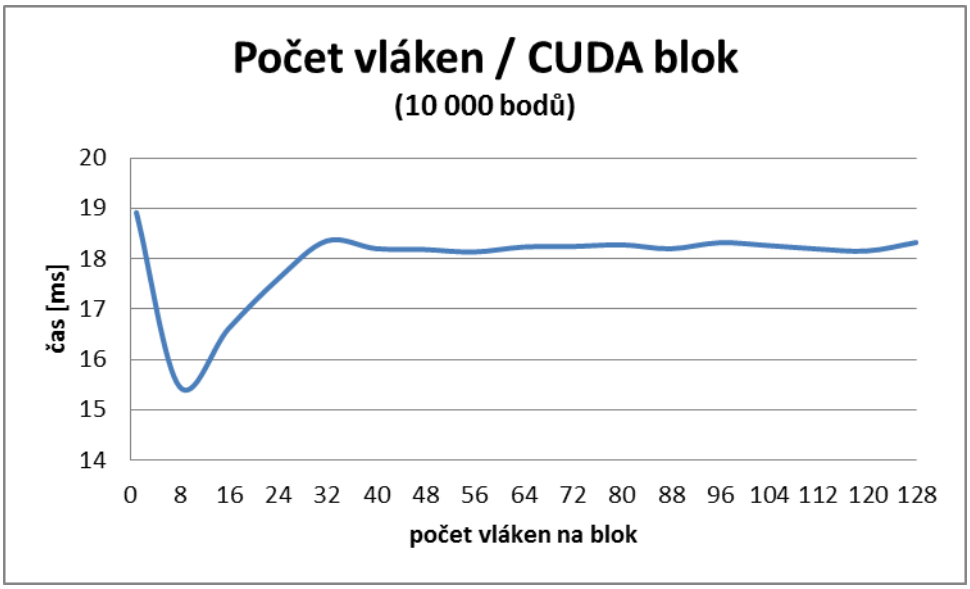
## B.2 Počet vláken na blok v CUDA (GPU)



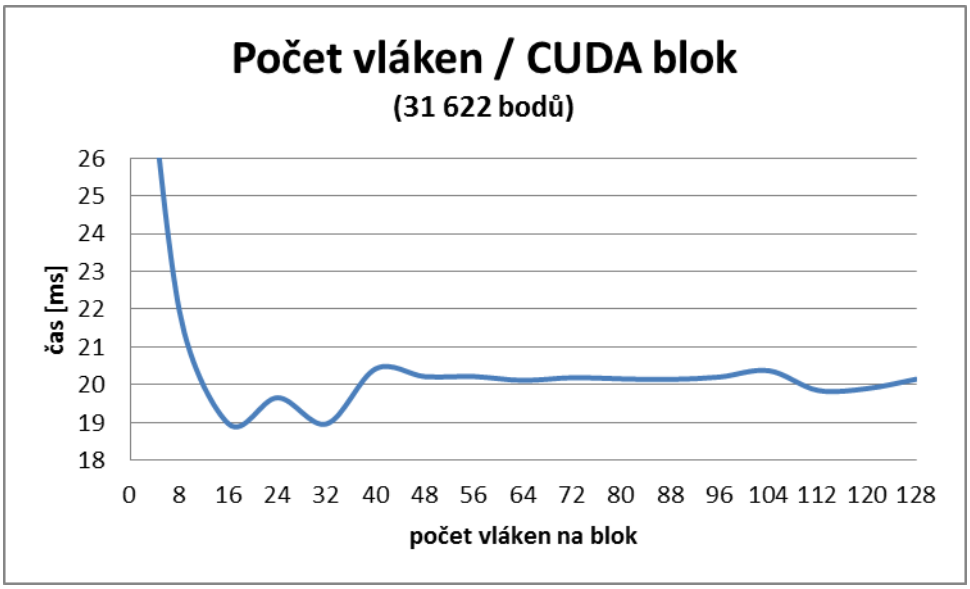

Graf 11.15: Časová náročnost v závislosti na počtu vláken na blok.



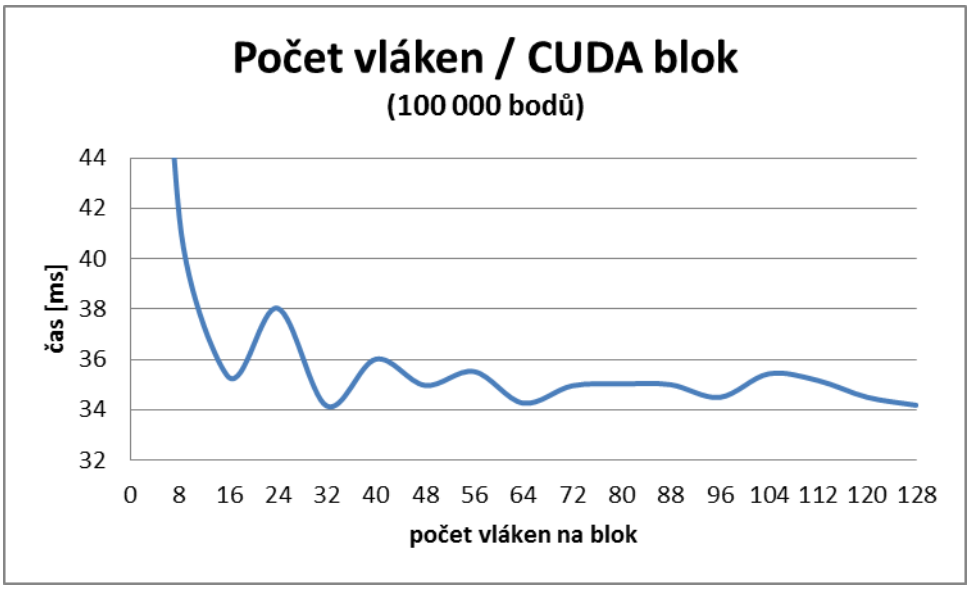

Graf 11.16: Časová náročnost v závislosti na počtu vláken na blok.



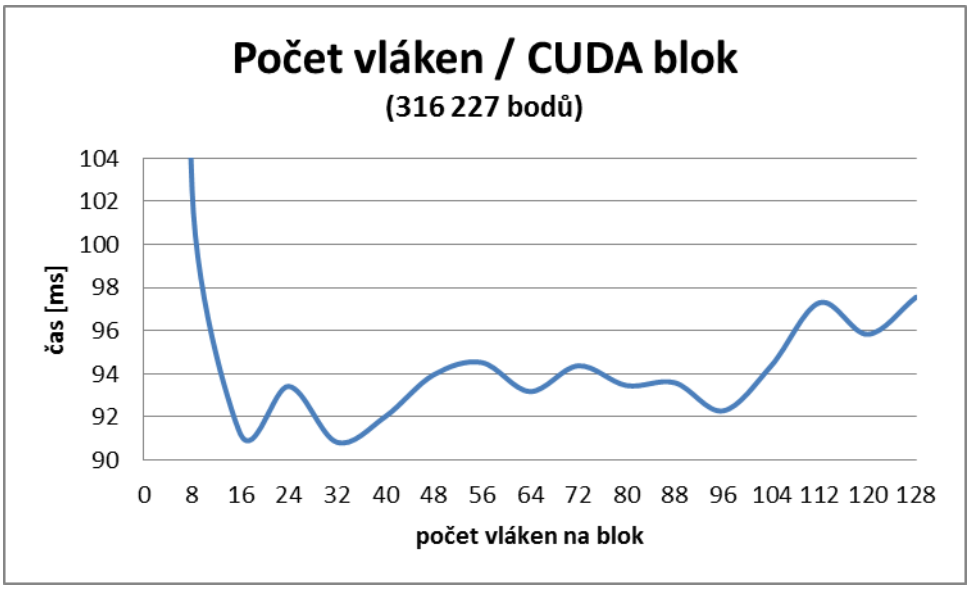

Graf 11.17: Časová náročnost v závislosti na počtu vláken na blok.



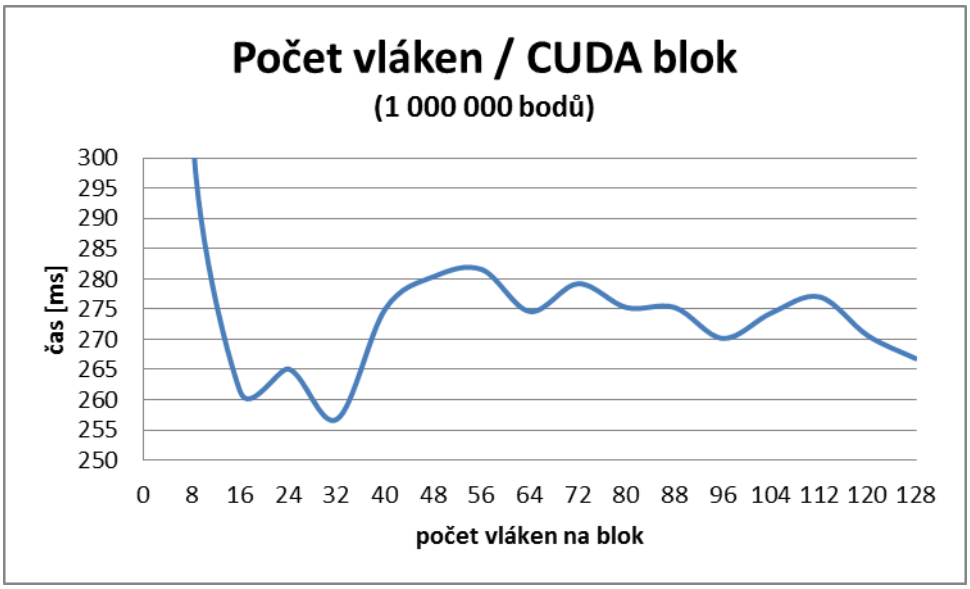

Graf 11.18: Časová náročnost v závislosti na počtu vláken na blok.




Graf 11.19: Časová náročnost v závislosti na počtu vláken na blok.



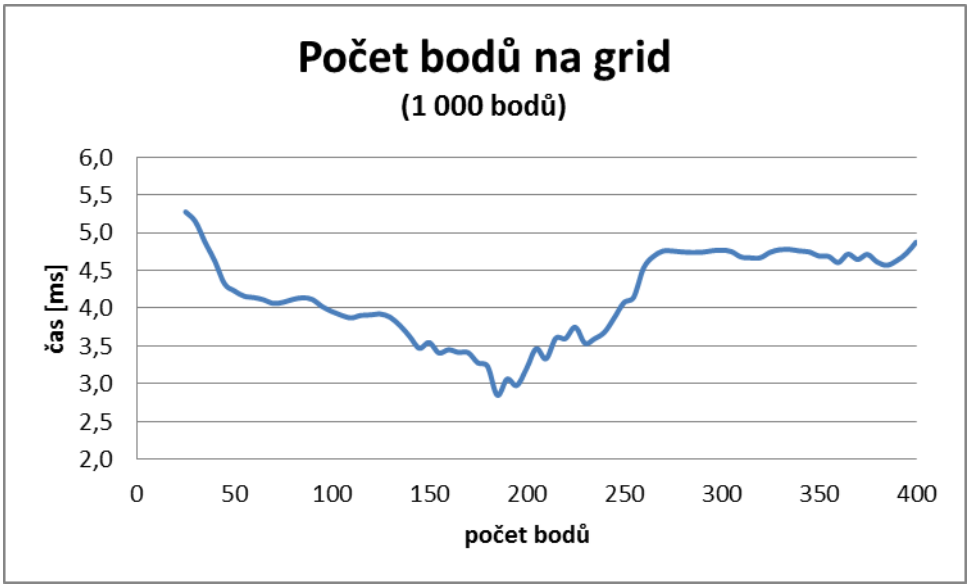

Graf 11.20: Časová náročnost v závislosti na počtu vláken na blok.



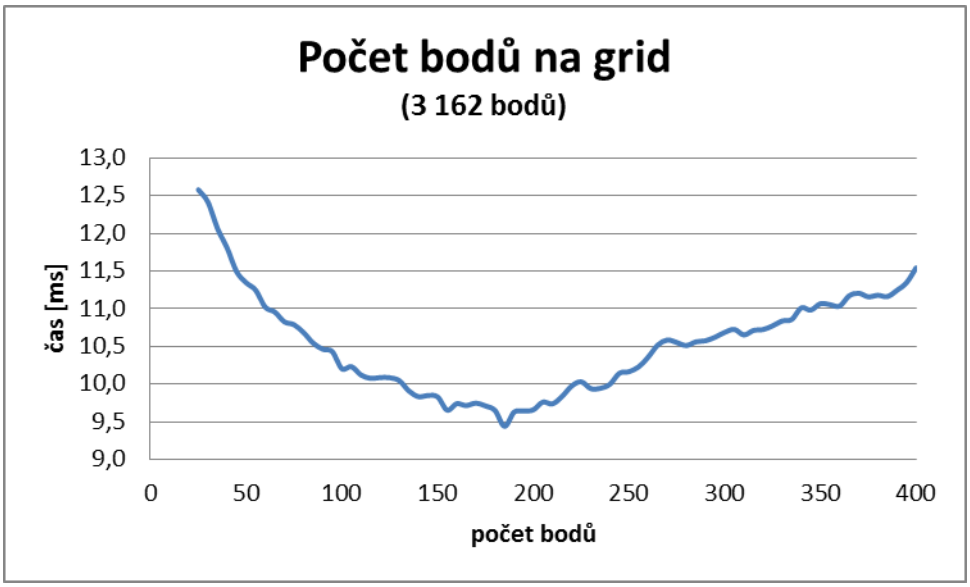

Graf 11.21: Časová náročnost v závislosti na počtu vláken na blok.

# C Grafy – tetrahedronizace

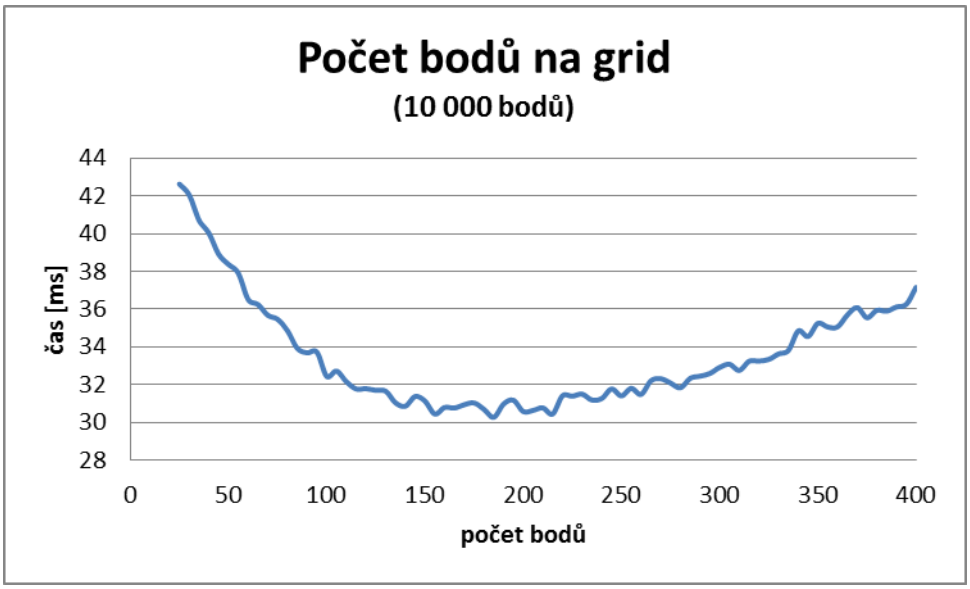
## C.1 Počet bodů na grid



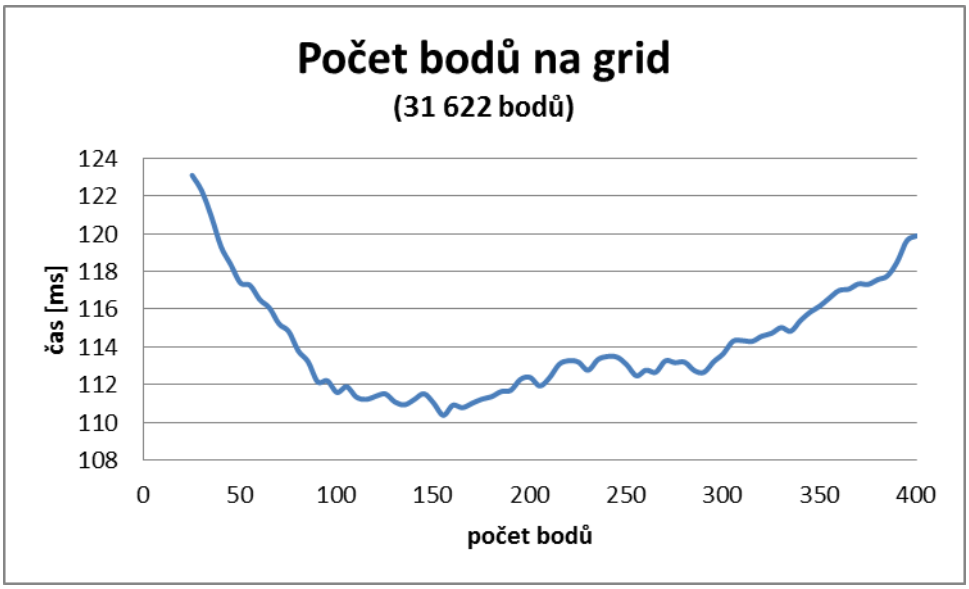

Graf 11.22: Časová náročnost v závislosti na počtu bodů na grid.



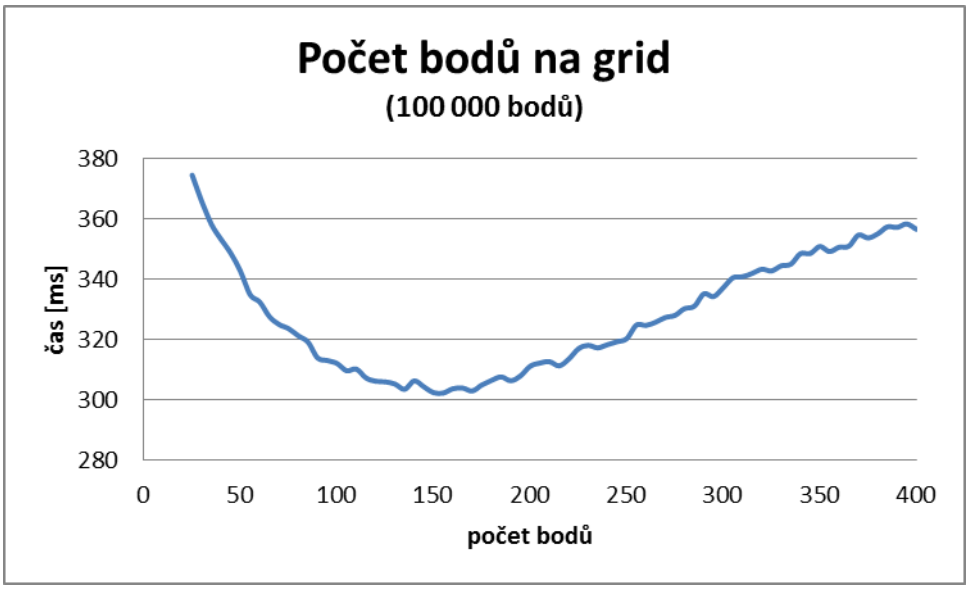

Graf 11.23: Časová náročnost v závislosti na počtu bodů na grid.



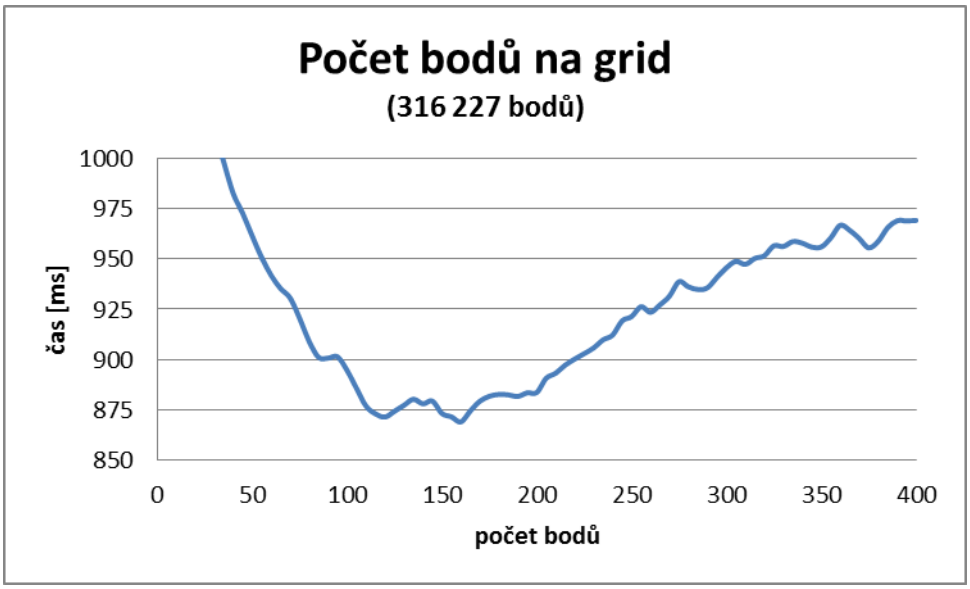

Graf 11.24: Časová náročnost v závislosti na počtu bodů na grid.



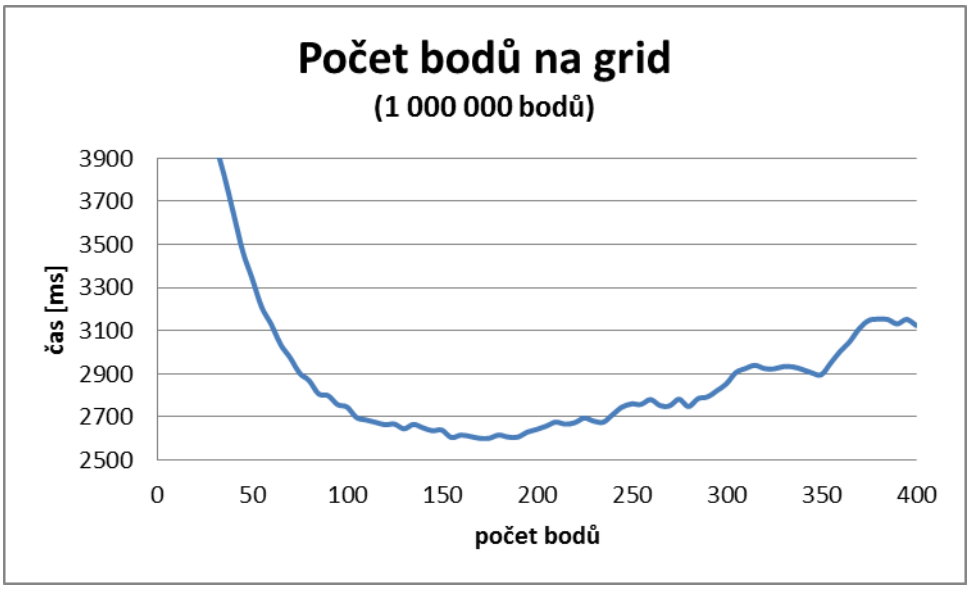

Graf 11.25: Časová náročnost v závislosti na počtu bodů na grid.



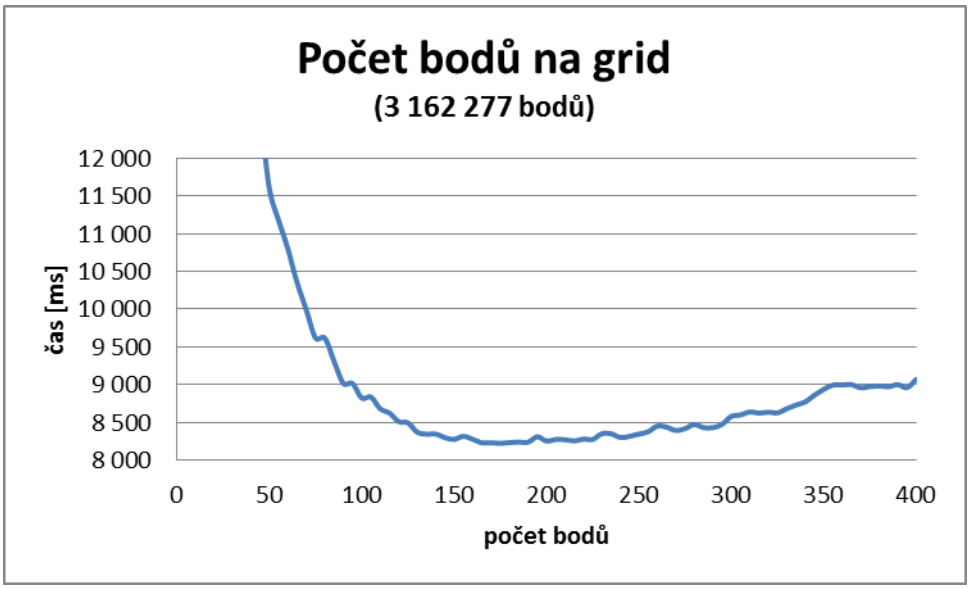

Graf 11.26: Časová náročnost v závislosti na počtu bodů na grid.



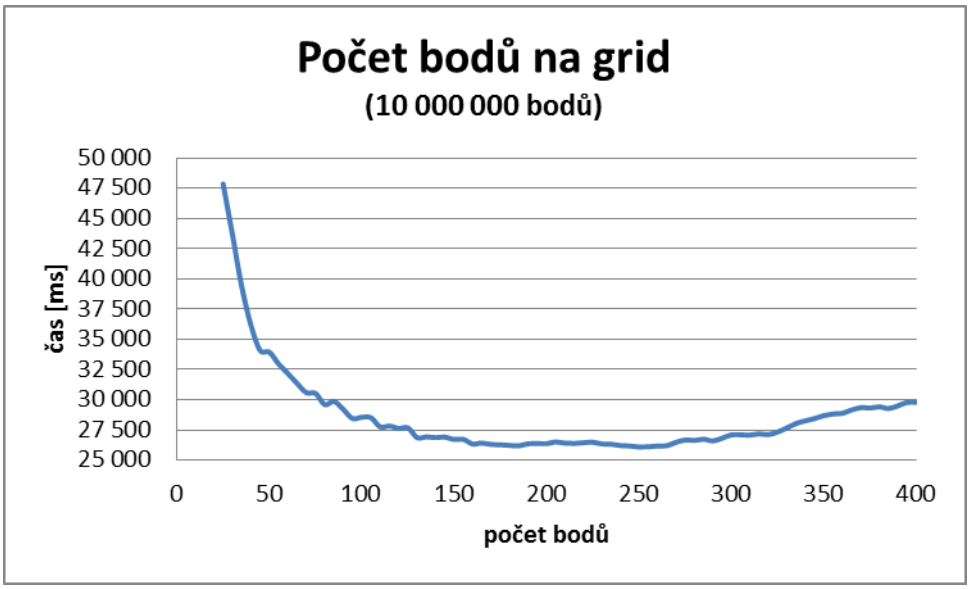

Graf 11.27: Časová náročnost v závislosti na počtu bodů na grid.




Graf 11.28: Časová náročnost v závislosti na počtu bodů na grid.




Graf 11.29: Časová náročnost v závislosti na počtu bodů na grid.




Graf 11.30: Časová náročnost v závislosti na počtu bodů na grid.