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

Diplomová práce

Delaunayova triangulace s omezením (CDT) v E^2 a E^3

Plzeň, 2006 Miroslav Fuksa

Constrained Delaunay triangulation (CDT) in E^3 and E^4

This thesis deals with problem of constrained Delaunay triangulation (CDT). CDT is a Delaunay triangulation constructed upon input points and input edges. This thesis describes principles of CDT in two dimensions (E^2). It presents an algorithm Maur which has been developed two years ago and not tested yet. This method was implemented and tested as a proof of functionality of this algorithm. Algorithm has been researched for an opportunity to use it as a compression algorithm for triangular meshes.

The second part of this thesis researches behavior of methods expanded to three dimensions. Three algorithms for constructing CDT have been described and two of them have been chosen for implementation. Although these methods do not successfully construct CDT they bring valuable experience and knowledge of developing CDT algorithms for three dimensions.

Obsah

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

V Plzni dne 18. 5. 2006

Miroslav Fuksa

1 Úvod

Tématem diplomové práce je constrained Delaunayova triangulace, zkráceně CDT. Jedná se o problém Delaunayovy triangulace, kde vstupem nejsou pouze body, ale i některé hrany nebo stěny. Triangulace vzniká propojením bodů v trojúhelníky. Triangulace je častým pojmem v mnoha technických oborech jako je například geodézie, grafické modelování nebo počítačová grafika. Uveďme si zde příklad použití triangulace v počítačové grafice.

Představme si, že máme nějaký terén, který potřebujeme vizualizovat. Vstupem je množina bodů naměřených v terénu. Každý bod je popsán souřadnicemi a naměřenou výškou. Pokud zobrazíme samotné body, výsledek nebude vypadat dobře. Druhé řešení je propojit body pomocí trojúhelníků a zobrazit vzniklé trojúhelníky. Toto řešení je samozřejmě lepší. Trojúhelníky navíc můžeme potáhnout nějakou texturou. Nebo můžeme jednoduše dopočítat výšku pomocí lineární interpolace i v místech, kde nejsou naměřené hodnoty. To bychom sice mohli udělat i z bodů, ale pro každý bod, který chceme pomocí interpolace vypočíst, bychom nejprve museli najít nejbližší body a pak nějakou funkcí dopočítat výšku. V případě triangulace stačí zjistit, v jakém trojúhelníku bod leží, a dopočíst výšku interpolací z jeho tří vrcholů. Vytvořit triangulaci je tedy dobrým řešením. Delaunayova triangulace je potom speciální typ triangulace, který má vhodné vlastnosti pro podobné aplikace.

Dále předpokládejme, že terén je naměřen na hranicích dvou států. Našim cílem je například zobrazit jenom jeden ze států. Je potřeba tedy druhý stát oříznout. Abychom mohli mezi dvěma body *a* a *b* provést řez, musí mezi body *a* a *b* vést hrana. Pokud ovšem triangulace tuto hranu neobsahuje, je potřeba ji do triangulace nějakým způsobem vnutit. K tomu lze použít CDT. Vstupem pro algoritmus CDT jsou tedy naměřené body a hrany tvořící hranici státu. Výstupem je triangulace nad vstupními body, která obsahuje zadané hrany.

Pro konstrukci CDT v prostoru E^2 existuje mnoho algoritmů. Jeden z nich byl navrhnut na katedře informatiky a výpočetní techniky fakulty aplikovaných věd v rámci doktorandského studia Pavla Maura. Tento algoritmus nebyl však implementován. Jeden z cílů diplomové práce je algoritmus doplnit o potřebné detaily, implementovat a otestovat.

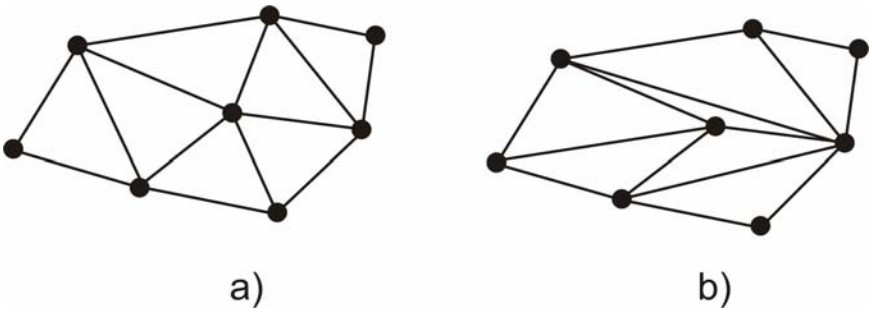
Dalším cílem diplomové práce je popsat možné použití metody Maur na kompresi trojúhelníkových sítí.

Součástí diplomové práce je také navržení a implementace rozšíření metod z prostoru E^2 do prostoru E^3. Pokud se E^2 varianta algoritmu Maur ukáže být funkční, je dalším postupem navrhnout jeho rozšíření do prostoru E^3. Pokud se ukáže algoritmus být použitelným v prostoru E^3, je zapotřebí jej implementovat. Pokud se neukáže být použitelným, má být prozkoumána problematika konstrukce CDT v E^3. Poté je cílem vybrat nějaké vhodné metody pracující v prostoru E^2 a pokusit se je rozšířit do prostoru E^3. Cílem nemusí být stoprocentně funkční metoda, ale charakterisitka chování metod v prostoru E^3. Jedna z možností je zvolit existující algoritmus na konstrukci CDT v E^3 a implementovat jej. Naprogramovaná metoda má být testována a má být popsáno chování metody.

2 Problematika Delaunayovy triangulace

2.1 Triangulace

V úvodu je stručně vysvětlen pojem triangulace a její použití. Jedná se o vytvoření trojúhelníkové sítě nad množinou vstupních bodů. Možností, jak takovou síť vytvořit, je mnoho. Různé sítě obsahující stejnou množinu bodů můžeme vidět na Obr. 2.1.



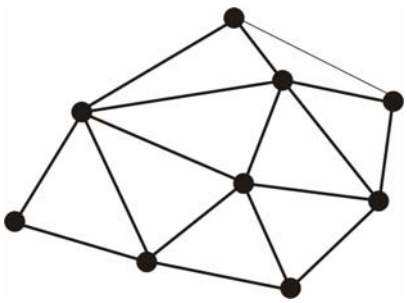
Obr. 2.1: Dvě možné varianty triangulace nad stejnou množinou bodů. Pro vytvoření byly použity dvě různé metody.

Nyní definujme triangulaci přesněji.

Definice 2-1, Triangulace:

> Nechť $G = (V, E)$ je graf, kde V je množina vrcholů a E je množina navzájem se neprotínajících hran určených vrcholy V. Graf G se nazývá triangulace, pokud E je maximální. [angla97a]

Definice říká, že při triangulaci jsou bodu propojeny hranami. To, že množina hran E je maximální, znamená, že v triangulaci nesmějí být „*díry*“, to znamená, že všechny vzniklé polygony jsou trojúhelníky. Zároveň z toho plyne, že do triangulace patří i hrany konvexní obálky. Na Obr. 2.2 je znázorněn graf, který k tomu, aby byl triangulací potřebuje přidat slabě vyznačenou hranu.



Obr. 2.2: Triangulace není úplná, pokud nezahrnuje všechny možné hrany včetně konvexní obálky. Bez hrany označené slabou čárou daný graf není triangulací.

Z výše uvedeného je zřejmý fakt, že triangulace je vždy konvexní. Tento poznatek se bude hodit při odvozování důkazů ohledně triangulace.

V následujícím textu je vysvětlen pojem Delaunayova triangulace.

2.2 Delaunayova triangulace

Na Obr. 2.1 jsou dvě různé triangulace sestrojené nad stejnou množinou bodů. Jedná se sice o dvě korektní triangulace, jejich praktické použití je však značně rozdílné. Triangulace b) obsahuje dlouhé a tenké trojúhelníky zatímco triangulace a) se skládá ze skoro rovnoramenných trojúhelníků. Dlouhé a tenké trojúhelníky mají několik zásadních nevýhod. Problém nastává při vykreslování a stínování takových trojúhelníků. Objekt sestávající se z tenkých trojúhelníků při vykreslení je vizuálně horší, než model z trojúhelníků rovnostranných. Stínování není pravidelné. Zároveň taková vizualizace zkresluje originální těleso, jelikož kvůli tenkým trojúhelníkům mohou vznikat ostré přechody, které ve skutečnosti v reálném objektu neexistují. Následující příklad demonstruje, kdy toto zkreslení může nastat. Při snímaní terénu vytváříme trojúhelníky dlouhé a tenké namísto rovnostranných. Tyto tenké trojúhelníky jsou vytvářeny ze vzdálenějších bodů než trojúhelníky rovnostranné. Předpokládejme celkem plynulý přechod výšky terénu. Čím vzdálenější body pro vytvoření trojúhelníku zvolíme, tím větší rozdíl může být v jejich výšce. Tím bude zároveň větší odchylka směru normál trojúhelníků z nich vytvořených. Může se tedy stát, že dva sousední trojúhelníky mají velkou odchylku normál i přes to, že skutečné normály v bodech uvnitř trojúhelníku (vypočtené například z funkce povrchu) jsou téměř stejné. Rozdíl normál trojúhelníků vznikl tím, že se do jejich výpočtu zohlednily body, které jsou relativně hodně vzdálené. Tím vznikají ostré přechody, které ve skutečnosti neexistují.

Druhým problémem dlouhých a tenkých trojúhelníků je interpolace. Hodnota získaná interpolací uvnitř takového trojúhelníku se ve většině případů odlišuje od skutečné hodnoty více než při použití rovnostranných trojúhelníků. Důvod je podobný jako v předchozím příkladu. Při použití dlouhých a tenkých trojúhelníků jsou voleny body více vzdálené. Vnitřní hodnoty se potom tedy dopočítávají z bodů, které neleží v okolí interpolovaného bodu. Je to tím, že vždy aspoň jeden bod dlouhého trojúhelníku je relativně hodně vzdálený. Přesto je tento bod do výpočtu zahrnut. Použití rovnostranných trojúhelníků tento problém řeší. Zde jsou interpolované hodnoty dopočítávány z bodů, které leží vzájemně blíže než body dlouhých trojúhelníků. Proto interpolovaná hodnota více odpovídá skutečné hodnotě.

Další nevýhodou použití dlouhých trojúhelníků jsou problémy numerické při výpočtech. I zde je většinou vhodné mít trojúhelníky co nejvíce rovnostranné.

Triangulace na Obr. 2.1 a) je tedy vhodnější pro předejití podobným problémům při řešení. Není náhodou, že tento obrázek znázorňuje právě Delaunayovu triangulaci. Jak je Delaunayova triangulace definovaná, je rozebráno v následujících odstavcích.

Delaunayova triangulace má tu vlastnost, že ze všech možných metod vytváří trojúhelníky nejbližší rovnostranným trojúhelníkům. Další vlastností je to, že Delaunayova triangulace maximalizuje minimální úhel. Minimální úhel je nejmenší úhel ze všech úhlů v trojúhelnících v triangulaci. Tento minimální úhel je větší při použití Delaunayovy triangulace než při použití jakékoliv jiné metody. Následující definice vystihuje hlavní vlastnosti Delaunayovy triangulace a zároveň je i částečným návodem, jak triangulaci konstruovat.

Definice 2-2, Delaunayova triangulace - hrany:

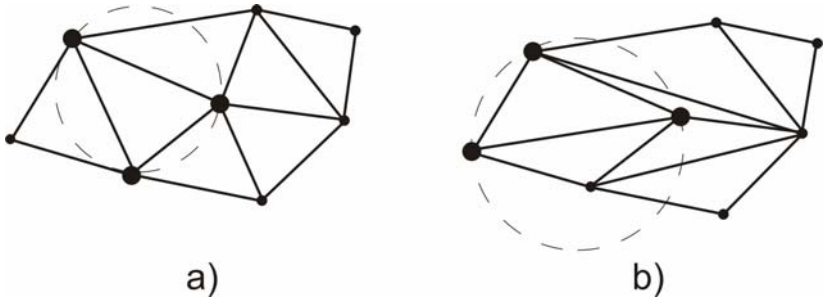
> Triangulace $\Delta = (V, E)$ je považována za Delaunayovu triangulaci, jestliže všechny hrany $ab \in E$ splňují vlastnost tzv. *prázdné opsané kružnice* (vzhledem k množině vrcholů V), kterou lze definovat následovně: existuje kružnice, která prochází vrcholy a, b taková, že všechny ostatní body z množiny V leží vně této kružnice. [angla97a]

Tuto definici lze formulovat i pro celé trojúhelníky.

Definice 2-3, Delaunayova triangulace - trojúhelníky:

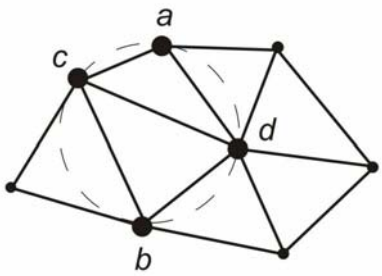
Pro každý trojúhelník z Delaunayovy triangulace $\Delta = (V, E)$ platí, že jemu opsaná kružnice neobsahuje žádné jiné body z V.

Tato definice je pro Delaunayovu triangulaci užitečnější. Pomocí ní pracuje většina algoritmů, a to tak, že testují právě prázdnou opsanou kružnici. Znázornění testu na prázdnou opsanou kružnici je na Obr. 2.3.



Obr. 2.3: Kružnice opsaná každému trojúhelníku v Delaunayově triangulaci neobsahuje žádný další bod z množiny bodů. Na obrázku b) není proto triangulace Delaunayovou triangulací. K tomu, aby bylo dokázáno, že triangulace a) je Delaunayova, je potřeba na prázdnou opsanou kružnici otestovat každý trojúhelník. Na obrázku je pro přehlednost znázorněn pouze jeden test.

Je zapotřebí ještě uvést, že v určitém singulárním případě je možné, aby na kružnici leželo více bodů z Delaunayovy triangulace. Tento případ nastává, pokud více než tři body v triangulaci leží na kružnici. Na Obr. 2.4 je tento případ znázorněn.



Obr. 2.4: Speciální případ, kdy 4 body leží na kružnici. V tomto případě může místo hrany *cd* být v triangulaci hrana *ab*. Obě řešení jsou Delaunayovou triangulací.

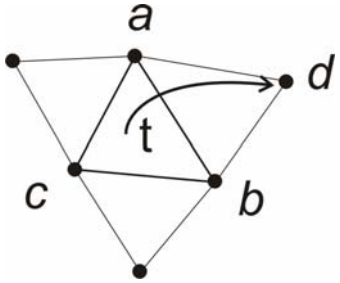
Delaunayova triangulace vytvořená nad vstupní množinou bodů má tu vlastnost, že pokud neobsahuje žádné singulární případy rozložení bodů, je vždy jednoznačná. To znamená, že neexistuje žádná jiná triangulace, která by splňovala Delaunayovo kritérium.

Delaunayova triangulace je duálním doplňkem Voroniových diagramů a lze ji použít pro jejich konstrukci.

Ke konstrukci Delaunayovy triangulace existuje mnoho algoritmů. Pro jejich bližší pochopení je potřeba se seznámit s následujícími poznatky. Nejprve si zaveďme pojem protější bod.

Definice 2-4, Protější bod:

Nechť *abc* je trojúhelník v triangulaci a *adb* sousední trojúhelník k trojúhelníku *abc*. Protější bod k trojúhelníku *abc* přes hranu *ab* je bod *d*. Situaci znázorňuje Obr. 2.5.



Obr. 2.5: Protější bod k trojúhelníku *abc* přes hranu *ab* je bod *d*.

Se znalostí pojmu protější bod je možné napsat následující definici, která popisuje lokálně optimální hranu podle Delaunayova kritéria.

Definice 2-4, Delaunayovská lokálně optimální hrana:

> 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*.

Hrana může být lokální optimální podle jakéhokoliv jiného kritéria. Pro naše účely bylo zvoleno Delaunayovo kriterium.

Při testování, zda je hrana lokálně optimální, je zapotřebí zjistit, zda se nachází protější bod uvnitř kružnice opsané bodům daného trojúhelníku. Jedna z možností, jak toto otestovat, je vypočítat střed a poloměr kružnice, sestrojit z těchto parametrů rovnici kružnice a poté testovat dosazením protějšího bodu, zda leží uvnitř nebo vně. Nerovnost, kterou musí testovaný bod splňovat, aby byl uvnitř, vypadá následovně:

$$(x - x_0)^2 + (y - y_0)^2 - r^2 \leq 0$$

Rov. 1: Podmínka bodu uvnitř kružnice

, kde x, y jsou souřadnice testovaného bodu, x_0, y_0 je střed kružnice a r poloměr. Pokud je nerovnice splněna, bod leží uvnitř kružnice. Problém je však výpočetně poměrně složitý. Existuje snazší varianta, která nahrazuje hledání parametrů kružnice a počítání hodnoty nerovnice. Bez odvození je zde uvedena výsledná forma. K otestování bodu stačí pouze vypočíst hodnotu determinantu

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

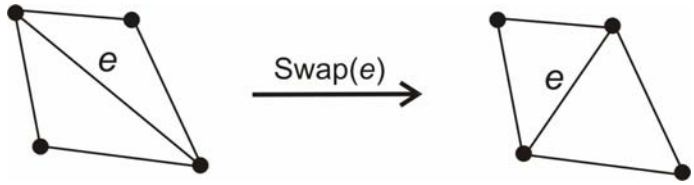
Rov. 2: Test bodu uvnitř kružnice

, kde a, b, c jsou body tvořící opsanou kružnici (indexy x,y označují jejich složky) a bod d je testovaný bod. Pokud je det > 0, leží bod uvnitř kružnice, pokud je det < 0, leží vně, a pokud det = 0, leží bod d na kružnici tvořené body a, b, c. Při výpočtu je důležitá orientace bodů. Tyto výsledky jsou platné pro body orientované proti směru hodinových ručiček. Při opačné orientaci vycházejí opačné výsledky. Existuje netriviální úprava, která převede tento determinant 4x4 na determinant 3x3. Upravený determinant vypadá následovně:

$$\det = \begin{vmatrix} a_x - d_x & a_y - d_y & (a_x - d_x)^2 + (a_y - d_y)^2 \\ b_x - d_x & b_y - d_y & (b_x - d_x)^2 + (b_y - d_y)^2 \\ c_x - d_x & c_y - d_y & (c_x - d_x)^2 + (c_y - d_y)^2 \end{vmatrix}.$$

Rov. 3: Test bodu uvnitř kružnice po úpravě

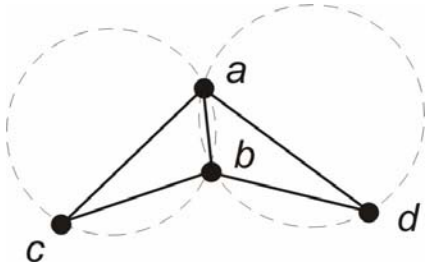
Takto se zjistí, zda bod leží uvnitř kružnice opsané trojúhelníku. Pokud testem zjistíme, že hrana je lokálně optimální vzhledem k jednomu trojúhelníku, označme jej *t*, je lokálně optimální i vzhledem k trojúhelníku sousedícího s trojúhelníkem *t* přes tuto hranu. Tento trojúhelník bude tedy splňovat test prázdné opsané kružnice vzhledem k bodům trojúhelníku *t*. Není potřeba tedy testovat hranu pro oba trojúhelníky. Pokud protější bod leží vně kružnice, je hrana lokálně optimální. Pokud však lokálně optimální není, je potřeba jí optimalizovat pomocí prohození. Při této operaci se změní struktura dvou sousedních trojúhelníků. Prohození se v anglické terminologii nazývá *flip* nebo také *swap*. Obr. 2.6 znázorňuje operaci prohození, jejímž argumentem je hrana *e*.



Obr. 2.6: Prohození hrany (flip, swap).

Pokud nějaká hrana není lokálně optimální a je na ní provedena operace prohození, hrana se stává lokálně optimální. Pokud nějaká hrana je lokálně optimální, neznamená to, že ve výsledné triangulaci musí být. Pokud totiž po prohození hrany *e* bude prohozena jiná hrana v témže trojúhelníku obsahujícím *e*, trojúhelník se opět změní a vzhledem k novému tvaru trojúhelníku a tedy i kružnice opsané je možné, že nová hrana lokálně optimální nebude. Dalo by se tedy říci, že lokální optimálnost je nutnou podmínkou k tomu, aby hrana byla ve výsledné triangulaci, ne však postačující.

Dva trojúhelníky, které dohromady tvoří nekonvexní polygon, nejdou prohodit. Je to tím, že diagonála by musela ležet vně trojúhelníků, což je nekorektní. Nikdy však nenastane případ, kdy by kružnice opsaná jednomu trojúhelníku obsahovala bod sousedního trojúhelníku. Je to tím, že kružnice jsou vypouklé vně od hrany. Situaci znázorňuje Obr. 2.7.



Obr. 2.7: Dva nekonvexní trojúhelníky. Jejich kružnice opsané nikdy nebudou obsahovat protější body.

2.3 Algoritmy konstrukce Delaunayovy triangulace

Algoritmů pro konstrukci Delaunayovy triangulace je mnoho. Algoritmy lze rozdělit na online algoritmy a ty, které online nejsou. Online znamená, že do hotové triangulace lze stále přidávat body. Vstupní množinu bodů není nutno znát předem. To je rozdíl oproti algoritmům, které online nejsou. Tam potřebujeme znát celou vstupní množinu ještě dříve,

než začneme algoritmus provádět. Termín *online algoritmus* je používán i u jiných algoritmů než jen u konstrukcí triangulace. Některé online algoritmy potřebují znát rozsah oblasti, do které budou body přidávány.

V následujícím textu je rozebráno několik algoritmů. Důvod, proč jsou zde rozebrány, je ten, že na jejich principu bude dále vysvětlena navazující problematika.

2.3.1 Lokální prohazování hran

Algoritmus prohazuje hrany, dokud nejsou všechny lokálně optimální. Tento postup lze použít i pro jiné triangulace než jen pro Delaunayovu. Kriterium lokálně optimální hrany by bylo potom jiné. Pro Delaunayovu triangulaci je tím kritériem právě lokálně optimální Delaunayova hrana, tedy prázdná opsaná kružnice.

Algoritmus by se dal zapsat v pseudokódu asi následovně:

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

(algoritmus 2-1: Lokální prohazování hran)

Co tedy algoritmus dělá? Vytvoří libovolnou triangulaci nad vstupní množinou[1]. Všechny hrany algoritmus zařadí do zásobníku a postupně testuje, zda jsou lokálně optimální.

Pokud nějaká hrana není lokálně optimální, provedeme na ní operaci prohození, hrana se tím stává lokálně optimální. Pokud nějaká hrana je lokálně optimální, neznamená to, že ve výsledné triangulaci musí být. Pokud po prohození hrany *e* bude prohozena jiná hrana v témže trojúhelníku, trojúhelník se změní a vzhledem k novému tvaru trojúhelníku a se změní i kružnice opsaná a je tedy možné, že hrana *e* lokálně optimální přestává být. Proto je potřeba po prohození hrany přidat do zásobníku i zbylé 4 hrany z dvojice trojúhelníků (pokud tam již nejsou).

Algoritmus takto legalizuje hrany, dokud se zásobník nevyprázdní. V takovém případě jsou všechny hrany lokálně optimální a tím je celá triangulace Delaunayova.

Tento algoritmus není online, protože potřebuje znát všechny body předem.

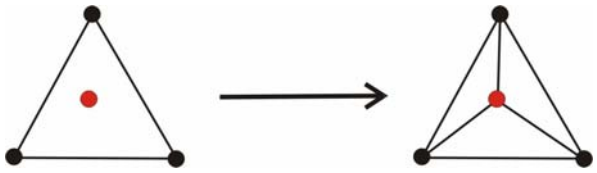
2.3.2 Inkrementální vkládání

Algoritmus inkrementálního vkládání je online algoritmem konstrukce DT. Jediné, co potřebujeme znát předem, je maximální možný rozsah souřadnic rozmístění bodů. Algoritmus pracuje tak, že na počátku vytvoří dostatečně velký trojúhelník z uměle přidaných bodů. Do vnitřku tohoto trojúhelníku musí patřit všechny příchozí body. Tři uměle přidané body musí splňovat podmínku takovou, že nesmějí ležet uvnitř žádné opsané kružnice nějaké hrany vzniklé triangulací. Teoreticky by měl mít velikost nekonečnou. Tento trojúhelník bude obsahovat celou triangulaci.

Do trojúhelníku se na počátku vloží první vstupní bod. Trojúhelník se rozdělí na tři části, jako je to na Obr. 2.8.

---

[1] Algoritmy na vytvoření libovolné triangulace zde rozebrány nebudou.



Obr. 2.8: Rozdělení trojúhelníku na 3 části. Na levém obrázku je znázorněn pouze vložený bod a na pravém je výsledek po rozdělení.

Zatím je vše jednoduché. Algoritmus však nebude o moc složitější. Budou se vkládat další body. Postup je stejný. Rozdělíme opět trojúhelník, ve kterém nový bod leží. Takto lze vytvářet triangulaci, která triangulací sice je, ale není Delaunayova. Proto, aby triangulace byla Delaunayova, je zapotřebí, aby po každém vložení bodu a obnovení triangulace byla provedena následující činnost. Vložený bod změnil trojúhelník, do kterého byl vložen. Proto je potřeba zkontrolovat jeho hrany, které teď náleží nově vzniklým trojúhelníkům. Situace je znázorněna na Obr. 2.9. Vlevo je triangulace skládající se pouze ze dvou trojúhelníků. Kružnice opsané trojúhelníkům neobsahují žádné jiné body triangulace. Po vložení bodu je na obrázku vpravo znázorněna nová konfigurace bodů a tedy i kružnic opsaných. Nyní obě kružnice obsahují bod sousedních trojúhelníků. Proto je potřeba hranu *e1* prohodit, aby se stala lokálně optimální.

Obr. 2.9: Obrázek znázorňuje, jak se mění triangulace po vložení bodu a jaký je vliv na splnění podmínky prázdné kružnice. Zatímco opsané kružnice levé triangulace neobsahují žádné sousední body, pravá triangulace s přidaným bodem tuto podmínku už nesplňuje. Na obrázku nejsou pro přehlednost vyznačeny kružnice pro testování hran e_2 a e_3.

Po každém vložení bodu je potřeba otestovat hrany nových trojúhelníků (na obrázku e_1, e_2, e_3). Pokud nejsou lokálně optimální, je potřeba je optimalizovat. Pokud prohození hrany provedeme, vytvoříme nové trojúhelníky, jejichž hrany nemusí být lokálně optimální. Je potřeba otestovat i ty a případně prohodit. Tento postup se rekurzivně aplikuje dokud nějaká hrana není lokálně optimální. Pro E^2 Delaunayovu triangulaci je tento postup vždy konečný a pokaždé vede k cíli.

Při implementaci se používají různé datové struktury pro urychlení vyhledávání, např. DAG. Jedná se o tzv. "pseudostrom" umožňující rychlé hledání trojúhelníků, v kterých se nachází nově vložený bod. Pomocí DAG lze nalézt trojúhelník v logaritmickém čase.

2.3.3 Přechod do vyšší dimenze

Algoritmus založený na přechodu do vyšší dimenze se v praxi moc nepoužívá. Je však velmi dobrou teoretickou pomůckou pro vznik dalších metod týkajících se Delaunayovy triangulace[2].

Podstata algoritmu je následující. Pokud jsou určitým způsobem namapovány body na kouli nebo na paraboloid, můžeme sestrojit v prostoru konvexní obálku tohoto tělesa určeného

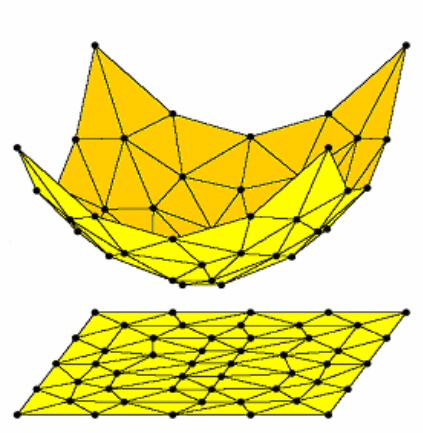
[2] Například algoritmus v [maur04].

namapovanými body. Pokud hrany tvořící konvexní obálku promítneme zpět do prostoru E^2, získáváme Delaunayovu triangulaci. Přesto, že tato teorie vypadá nepoužitelně a možná zprvu nepředstavitelně, je v dalších kapitolách vidět její praktické použití při odvozování zajímavých teorií.

V následujících odstavcích je problém rozebrán podrobněji. Mějme vstupní množinu bodů V složenou z bodů $v_i[x_i, y_i]$. Tyto body nyní promítneme na paraboloid. Operace promítnutí je následující. Pro tuto množinu bodů v E^2 dopočteme třetí souřadnici z následujícím způsobem: $z_i = x_i^2 + y_i^2$. Vzniklé body označme v_i^+ a množinu těchto bodů jako V^+. Operaci lze matematicky formulovat následovně:

$$v_i[x_i, y_i] => v_i^+[x_i, y_i, x_i^2 + y_i^2].$$

Body $v_i^+[x_i, y_i, z_i]$ leží na paraboloidu daném implicitní funkcí $0 = x^2 + y^2 - z^2$. Situaci znázorňuje Obr. 2.10.



Obr. 2.10: Body z prostoru E^2 namapované na paraboloid. Nad paraboloidem je vytvořena konvexní obálka, která je poté promítnuta do E^2, kde vytváří Delaunayovu triangulaci. Obrázek převzatý z [shew03a].

Dalším krokem je vytvořit v prostoru konvexní obálku těchto bodů. Tato konvexní obálka se skládá z trojúhelníkových ploch. Hrany, které tvoří stěny konvexní obálky, opět promítneme zpět do prostoru E^2. Operace zpětného promítnutí je následující. Pro každou hranu mezi v_i^+ a v_j^+ vytvoříme hranu mezi odpovídajícími si body v_i a v_j z množiny V.

Při zpětném promítání hran je však potřeba promítat jen hrany spodní části konvexní obálky. Vzniklá konvexní obálka totiž obsahuje i plochy uzavírající body shora (na obrázku pro přehlednost zakresleny nebyly). Tyto hrany je potřeba vyloučit. Vzniklé hrany v E^2 a body z množiny V tvoří Delaunayovu triangulaci.

Paraboloid, na který se body mapují, může mít svůj střed posunutý o libovolnou vzdálenost. Obecnější rovnice paraboloidu by vypadala následovně:

$$0 = (x-x_0)^2 + (y-y_0)^2 - z^2 - m\,x - n\,y - c$$

Rov. 2-4: Obecná rovnice paraboloidu

, kde $[x_0, y_0]$ je střed paraboloidu, c je libovolná konstanta určující posunutí středu paraboloidu ve směru osy z, m a n jsou konstanty určující naklonění paraboloidu. Konvexní obálka bodů namapovaných na paraboloid nezáleží na posunutí středu paraboloidu vůči vstupním bodům. Stručný důkaz tohoto tvrzení by byl asi následující: Delaunayova triangulace je nezávislá na středu souřadné soustavy. Znamená to, že výsledné hrany budou stejné při různém posunutí všech bodů o jakoukoliv vzdálenost. Delaunayova triangulace je pro nesingulární případy jednoznačná (singulární případy neuvažujme). Proto pro dvě různé,

navzájem posunuté množiny bodů musí být stejné i hrany konvexní obálky. To proto, že přímo korespondují s Delaunayovou triangulací v E^2, která je neměnná při libovolném posunutí v ose XY. Znamená to tedy, že triangulace může být namapována na jakýkoliv paraboloid. Pokaždé bude výsledek přechodu do vyšší dimenze stejný.

Algoritmus není v praxi příliš používán. Jeho použitelnost bude vidět v této diplomové práci při teoretickém odvozování.

2.4 Jiné triangulace

Jak jsme již mohli vidět, existují i jiné typy triangulací než Delaunayova. Jen stručně se o některých zmíníme.

Triangulace s minimální váhou (Minimum weight triangulation, MWT)

Minimalizuje celkovou délku všech hran v triangulaci. Při konstrukci je potřeba sestrojit všechny možné triangulace a vybrat z nich tu s minimální celkovou délkou hran. Pro větší počet vrcholů je časově velmi náročná. Metoda má exponenciální složitost. Lepší algoritmus na konstrukci není znám a není známo, zda je možné problém řešit v polynomiálním čase.

Greedy (žravá, hltavá) triangulace (GT)

Tato triangulace se skládá z nejkratších hran v triangulaci. Je aproximací MWT v tom smyslu, že se celkovou délkou hran k MWT blíží. Je definována algoritmem konstrukce.

Její složitost je narozdíl od MWT nižší. Při použití brutální síly složitost dosahuje sice $O(N^3)$, ale rychlejší algoritmy mají složitost $O(N^2 \log N)$. Pro speciální případ rovnoměrně rozložených dat může dosáhnout i výpočetní složitost $O(N)$. Greedy triangulace se konstruuje tak, že se vytvoří všechny možné hrany (samozřejmě se budou křížit), seřadí se podle délky a poté se vkládají do výsledné triangulace od nejkratší k delším tak dlouho, dokud není triangulace úplná. Hltavá se nazývá proto, že jak algoritmus jednou hranu vloží, již ji nemůže vyjmout. To znamená, že svá rozhodnutí nemění a nevrací se. Postupně takto „hltá“ hrany, dokud nevytvoří triangulaci.

Datově závislé triangulace

Triangulace berou v potaz i jiné vlastnosti než jen vzdálenost bodů. Touto vlastností bývá často i výška bodů nebo jakákoliv jiná hodnota v bodech naměřená. Pro každý trojúhelník například používá jako kritérium jednak vzdálenost bodů (jako u Delaunayovy triangulace), jednak odchylka normál dvou trojúhelníků. Díky tomu zachovávají tyto triangulace ostré výškové rozdíly tím, že na ostrých přechodech se po triangulaci vyskytnou tenké trojúhelníky kopírující vrstevnici terénu.

2.5 Constrained Delaunay triangulation (CDT)

Constrained Delaunayova triangulace (zkráceně CDT) je česky nazývána Delaunayova triangulace s omezením (v anglické terminologii další používaný termín je Restricted Delaunay Triangulation). Tato triangulace pracuje tak, že vytváří Delaunayovu triangulaci ze vstupní množiny bodů a hran. Některé hrany v triangulaci jsou tzv. vnuceny, tj. triangulace je musí obsahovat. Tyto vstupní hrany se nazývají povinné nebo vnucené. To, že můžeme vnutit do triangulace určité hrany, má mnoho využití. Jedním z nich je například triangulace nekonvexního tvaru při zachování vnější nekonvexní obálky. Delaunayova triangulace by obsahovala i konvexní obálku vstupních bodů. Při použití CDT můžeme definovat hrany obálky jako povinné. Z výsledného CDT potom můžeme oříznout vnější část mimo povinné

hrany a tím získáváme opět nekonvexní tvar. Další využití je například při modelování terénu, kde můžeme pomocí hran vnutit i trojúhelníky, které nejsou Delaunayovy.

Bude každá vnucená hrana Delaunayova? Nebude. Tento případ nastane pokaždé, když má být vnucena hrana, která není v Delaunayově triangulaci. To proto, že Delaunayova triangulace je při nesingulárním rozložení bodů vždy unikátní. Jakákoliv jiná triangulace nad stejnou vstupní množinou už není Delaunayova. Takže vzniklá triangulace nebude splňovat Delaunayovo kritérium. Proto je zapotřebí definovat kritérium CDT.

Nejprve následuje definice viditelnosti, která bude použita v definici CDT.

Definice 2-5, Viditelnost:

> Dva vrcholy *a* a *b* v grafu *G = (V, E)* jsou navzájem viditelné, pokud přímka mezi *a* a b neprotíná žádnou hranu z *E*.

Definice popisuje viditelnost tak, jak je intuitivně chápána. Body na sebe vidí, pokud jim v tom nebrání nějaká hrana. Další definice popisuje CDT.

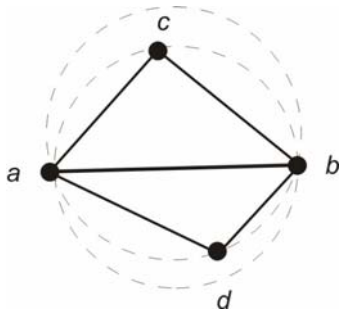
Definice 2-5, CDT:

> Nechť graf *G = (V, E)* je planární graf takový, že $E \neq \emptyset$. Triangulace $\Delta = (V, E \cup E')$ je CDT grafu *G*, pokud všechny hrany $ab \in E'$ jsou takové, že body *a, b* jsou navzájem viditelné v grafu *G,* a hrana *ab* splňuje podmínku prázdné opsané kružnice (definice 2-2) vzhledem k vrcholům viditelných z *a* a *b* v grafu *G* [angla97a].

Definice je na první pohled trochu komplikovaná. Hrany, které jsou do triangulace vnuceny, označuje definice množinou hran *E*. Tyto hrany jsou vstupem stejně tak jako body. Hrany *E'* jsou hrany, které jsou přidány v rámci procesu triangulace. Definice říká, že všechny hrany nevynucené z množiny *E'* musí splňovat podmínku, že jejich vrcholy jsou navzájem viditelné. To znamená, že tyto hrany se nesmějí křížit s hranami, které jsou vynucené jako vstupní hrany. Druhá část definice říká, že při posuzování Delaunayova kritéria o hranách z definice 2-2 jsou uvažovány pouze body, které jsou viditelné jak z *a* tak i z *b* s ohledem na vnucené hrany z grafu *G*. To znamená, že CDT může porušovat kriterium prázdné opsané kružnice hrany, pokud body, které do kružnice spadají, nejsou viditelné z bodů tvořících hranu, protože jim v tom brání vnucená hrana.

Tuto definici je možné použít opět i pro trojúhelníky. Podmínka říká, že kružnice opsaná trojúhelníku nesmí obsahovat žádný z bodů viditelných z vnitřku trojúhelníku.

Na Obr. 2.11 je znázorněna triangulace s vnucenou hranou *ab*. Tato triangulace nesplňuje podmínky Delaunayovy triangulace. Hrana *ab* je však vnucená, takže i přes to, že kružnice opsaná trojúhelníku *abc* obsahuje bod *d*, tento bod neporušuje kriterium CDT, protože není z trojúhelníku *abc* viditelný. Stejná podmínka platí i pro trojúhelník *abd*.



Obr. 2.11: Jednoduchý příklad vnucené hrany *ab*. Trojúhelníky *abc* a *abd* nejsou sice Delaunayovské, protože jejich opsané kružnice obsahují protější body. Přesto splňují podmínku CDT, protože trojúhelníky na sebe přes vnucenou hranu *ab* navzájem nevidí.

CDT může být tvořeno pouze hranami, které se nesmí navzájem křížit. Takto je definováno CDT v prostoru E^2. V následující kapitole bude vysvětlen pojem regulární triangulace.

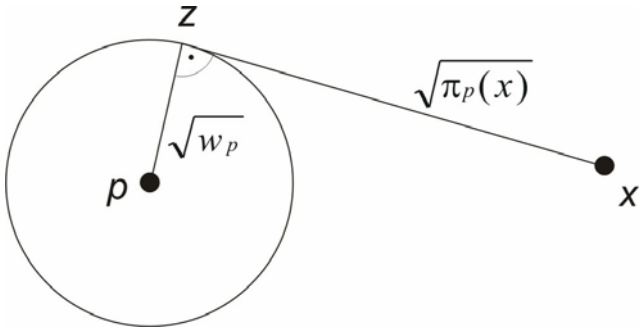
2.6 Regulární triangulace

Regulární triangulace je zobecněním Delaunayovy triangulace. Rozdíl oproti ní je v tom, že se vzdálenost mezi body nepočítá jako eukleidovská vzdálenost, ale do výpočtu jsou zahrnuty i váhy bodů. Způsob vypočtení vzdálenosti uvádí následující definice.

Definice2-6, Power distance:

Mějme množinu bodů $S \subset \boldsymbol{R^2} \times \boldsymbol{R}$, kde každému bodu $p = (x1,x2) \in S$ je přiřazena váha w_p. Bod s pozitivní váhou může být interpretován jako kružnice se středem p a poloměrem $\sqrt{w_p}$. Pro každý vážený bod p definujme power distance z bodu $x \in \boldsymbol{R^2}$ do p jako $\pi_p(x) = |x_p|^2 - w_p$, kde $|xp|$ je Euklidovská vzdálenost z bodu x do bodu p. [maur04]

Z definice vyplývá několik následujících poznatků. Vzdálenost nazývaná *power distance* se měří vždy z bodu, označme jej p, který má přiřazenu určitou váhu, k nějakému bodu v prostoru $\boldsymbol{R^2}$(tj. bod bez váhy). Bod p má k ostatním bodům tím menší power distance, čím větší je jeho váha. Situaci je možné přiblížit pomocí kružnice opsané bodu p, jak je znázorněno na Obr. 2.12.



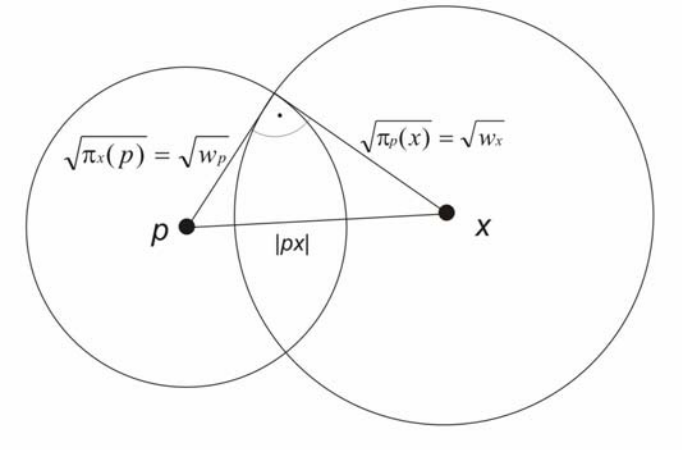
Obr. 2.12: Power distance bodu *p* od bodu *x* je druhá mocnina tečny eukleidovské vzdálenosti bodu *x* od tečného bodu *z* ke kružnici se středem *p* a poloměrem $\sqrt{wp}$.

Následující definice hovoří o tom, co jsou to ortogonální body .

Definice 2-7, Ortogonalita:

Dva vážené body p a x jsou ortogonální, pokud $|p_x|^2 = w_p + w_x$. Z toho vyplývá, že $\pi_x(p) = w_p$ a $\pi_p(x) = w_x$. Nechť tři vážené body tvoří trojúhelník A. Potom existuje jedinečný vážený bod z takový, že z je ortogonální ke všem váženým bodům z trojúhelníku A. Tento bod se nazývá ortogonální střed trojúhelníku A. [maur04]

Znázornění významu je ortogonality dvou bodů je na Obr. 2.13.



Obr. 2.13: Body *p* a *x* jsou navzájem ortogonální.

Pro ortogonální střed platí následující vlastnost. Pokud váhy bodů v trojúhelníku budou nulové, potom ortogonální střed je středem opsané kružnice. Zatím byla definována pouze pravidla pro výpočet vzdálenosti v regulárních triangulacích. Nyní následuje definice toho, co regulární triangulace je.

Definice 2-8, Regulární triangulace:

> Trojúhelník T je globálně regulární, pokud $\pi_z(q) > w_q$ pro všechny body $q \in S\text{-}\{T\}$. Množina globálně regulárních trojúhelníků tvoří regulární triangulaci.

Co tato definice znamená? Popisuje regulární triangulaci tak, že ortogonální středy trojúhelníků musí splňovat nějakou podmínku vůči všem ostatním bodům triangulace kromě bodů trojúhelníku. Pokud by všechny váhy bodů v triangulaci byly nulové, potom by se jednalo o testování, zda nějaký bod neleží uvnitř kružnice opsané a tím i o Delaunayovu podmínku. Taková triangulace by byla Delaunayova. Regulární triangulace je tedy zobecněním triangulace Delaunayovy.

Stejně tak jako Delaunayova triangulace má svou duální podobu ve Voroniových diagramech, tak i regulární triangulace má svou duální obdobu v power diagramech.

Konstrukce regulárních triangulací je velmi podobná konstrukcím Delaunayových triangulací. I zde se používá pojem lokálně regulární hrana.

Definice 2-9, Lokálně regulární hrana:

> Nechť *ab* je hrana triangulace $\Delta = (V, E)$ společná dvěma trojúhelníkům *abc* a *abd*. Nechť *z* je ortogonální střed trojúhelníku *abc*. Potom hrana *ab* je lokálně regulární pokud splňuje podmínku, že $\pi_z(d) > w_d$.

Jedná se tedy o definici podobnou Delaunayově lokálně optimální hraně (definice 2-4), jen je použité jiné kritérium. Lokálně regulární hrana ve výsledné triangulaci být nemusí. Tato podmínka je opět pouze nutnou podmínkou, ne postačující. Pokud nějaká hrana není lokálně regulární, lze ji optimalizovat pomocí operace prohození (viz. kapitola 2.2). Tato hrana se potom stává lokálně regulární.

Ke zjištění, zda je hrana regulárně optimální, se používá podobný postup jako u Delaunayova testu. Spočtením determinantu Rov. 5 zjistíme, zda testovaný bod splňuje pravidlo regulární optimality.

$$\det = \begin{vmatrix} a_x & a_y & a_x^2 + a_y^2 - w_a & 1 \\ b_x & b_y & b_x^2 + b_y^2 - w_b & 1 \\ c_x & c_y & c_x^2 + c_y^2 - w_c & 1 \\ d_x & d_y & d_x^2 + \mathrm{d}_y^2 - w_d & 1 \end{vmatrix}$$

Rov. 5: Test regulární hrany

V determinantu *a*, *b*, *c* jsou body tvořící trojúhelník, bod *d* je bod, který má být testován na splnění kritéria, a *w* jsou váhy jednotlivých bodů. Pokud je det > 0, nesplňuje bod kritérium lokálně regulárního bodu. Pokud det < 0, podmínka je splněna. Pokud je výsledek roven nule, potom se jedná o singulární případ. V případě Delaunayově triangulace tento případ znamená, že tyto 4 body leží na kružnici, a je v takovém případě jedno, jakou konfiguraci hrana bude mít. Totéž platí i pro regulární triangulaci. Výsledek ovlivňuje i orientace vstupních bodů stejným způsobem jako pro determinant Rov. 2.

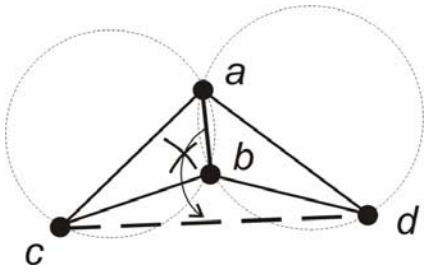
Při srovnání determinantů Rov. 2 a Rov. 5 je zřejmé, že jediný rozdíl je ve třetím sloupci, kde jsou navíc odečteny váhy bodů. Regulární triangulaci si lze představit stejně jako mapování bodů na paraboloid podobně jako v kapitole 2.3.3. Problém se dá nahradit problémem vytvoření konvexní obálky nad body vyzdvihnutými na paraboloid. Mapovací funkce pro regulární triangulaci by vypadala následovně:

$$v_i[x_i, y_i] \Rightarrow v_i^+[x_i, y_{i,}, x_i^2 + y_i^2 - w].$$

Rov. 6: Mapovací funkce regulární triangulace

Jediný rozdíl oproti mapovací funkci pro Delaunayovu triangulaci je v odečtení váhy od třetí souřadnice. Třetí souřadnici označme *z*. Může nastat situace, kdy bod bude mít podstatně nižší váhu než okolní body. Potom je možné, že se bod v konvexní obálce vůbec neobjeví. Takový bod se nazývá *redundantní bod*. Toto je rozdíl oproti Delaunayově triangulaci, kdy triangulaci tvoří vždy všechny vstupní body. I zde je vidět, že pokud všechny váhy bodů budou rovny nule, potom jde o problém mapování bodů „přesně“ na paraboloid, neboli o konstrukci Delaunayovy triangulace.

Konstrukce regulární triangulace je tedy podobná jako konstrukce Delaunayovy triangulace s tím rozdílem, že determinant, podle kterého se počítá lokální optimálnost hrany, je odlišný. Jsou zde však i jiné rozdíly. Při použití algoritmu lokálního prohazování hran se může stát, že algoritmus dojde do situace, kdy nemůže prohodit hranu. Situace, kdy toto může nastat, je znázorněna na Obr. 2.14. Jsou na něm dva trojúhelníky v nekonvexním tvaru. Při testu Delaunayovy podmínky nemůže nikdy dojít k prohození hrany, jak bylo popsáno v kapitole 2.2. Je to tím, že kružnice vždy směřují pryč od hrany. Tyto kružnice jsou znázorněny na obrázku. Při použití regulární triangulace tento případ nastat může. Stačí, aby bod *c* měl takovou váhu (relativně vysokou), že poruší podmínku pro trojúhelník *adb*.



Obr. 2.14: Dva trojúhelníky *abc* a *adb*. Kružnice jsou kružnice opsané. Pro Delaunayovu triangulaci je hrana *ab* vždy lokálně optimální. Nemusí však už být lokálně regulární, ale přes to nemůže být prohozena za hranu *cd* kvůli tvaru trojúhelníků.

V takovém případě nastává problém, protože hrana nemůže být kvůli nekonvexnímu tvaru prohozena. Problém se řeší pomocí zásobníku. Pokud hrana nemůže být prohozena, zařadí se do zásobníku a zpracuje se později. Problém se dá řešit pomocí inkrementální metody. Při inkrementální metodě je zaručeno, že algoritmus proběhne v pořádku.

V následující kapitole budou probrány algoritmy konstrukce CDT v E^2.

3 Existující algoritmy na CDT

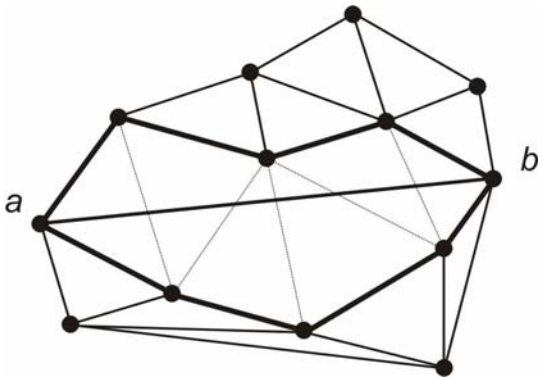
V této kapitole budou popsány algoritmy pro konstrukci CDT.

3.1 Anglada

Algoritmus, který je zde vysvětlen, je možné nelézt v [angla97a]. Algoritmus je online, podporuje průběžné vkládání bodů a hran. Popis omezíme pouze na vnucení hrany do hotové triangulace. Následujícím postupem lze tedy vnutit hranu do hotové Delaunayovy triangulace.

Nalezení zasažené oblasti

Algoritmus nalezne všechny hrany, které protíná vnucená hrana *ab*. Takových hran může být libovolné množství, záleží na délce hrany. Obr. 3.1 znázorňuje triangulaci, do níž má být vnucena hrana. Všechny protnuté hrany mají být z triangulace odstraněny.



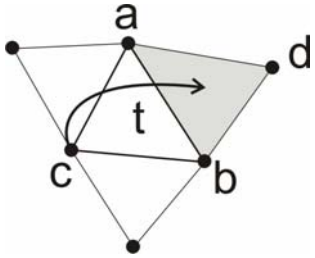
Obr. 3.1: Vnucení hrany *ab* do triangulace vymaže všechny hrany, které protíná. Silně je vyznačena zasažená oblast.

Oblast, která hranu ohraničuje, se nazývá *zasažená oblast*. Skládá se ze všech trojúhelníků, které jsou hranou protnuty. Existuje několik způsobů, jak takové trojúhelníky s protnutými hranami efektivně nalézt.

Existuje postup, který je také uveden v [angla97a]. Tento algoritmus M. V. Anglady, hledá protnuté trojúhelníky systémem procházení po sousedech směrem od bodu *a* do bodu *b*.

Na obrázku jsou to všechny hrany protnuté hranou *ab*. Princip hledání oblasti je následující. Algoritmus vychází z bodu *a* a prohledává ve směrem k *b*. Postupně prochází přes trojúhelníky, které protíná hrana. Nakonec skončí v trojúhelníku obsahující bod *b*. Všechny prošlé trojúhelníky postupně ukládá do seznamu.

K pochopení algoritmu je zapotřebí vysvětlit funkci `ProtějšíTrojúhelník(t,c)`. Princip je podobný jako u hledání protějšího bodu. Protější trojúhelník k bodu *c* v trojúhelníku *t* je sousední trojúhelník, který sdílí hranu *ab* s trojúhelníkem *t*, která leží naproti bodu *c*. Ukázka je znázorněna na Obr. 3.2.



Obr. 3.2: Protější trojúhelník.

Algoritmus přechází z bodu na bod kolem hrany až do bodu `b` hrany `ab`. Pokaždé vstoupí do všech bodů a všech trojúhelníků zasažené oblasti. Při průchodu ruší trojúhelníky, kterými projde. Druhá možnost je ukládat tyto trojúhelníky do seznamu.

Tímto postupem vznikají další dva seznamy, seznam vrcholů horní zasažené oblasti (P_U) a seznam vrcholů dolní zasažené oblasti (P_L). Zde je potřeba říci, že algoritmus vnitřně počítá testy na body horní a dolní pomocí orientačních testů. Na Obr. 3.1 horní znamená *vlevo od hrany* a dolní *vpravo od hrany ab* vzhledem k bodu *a*.

Orientační test se provádí spočtením následujícího determinantu Rov. 7.

$$\det = \begin{vmatrix} b_x & b_y \\ a_x & a_y \end{vmatrix}$$

Rov. 7: Orientační test

, kde $a=[a_x, a_y]$ a $b=[b_x, b_y]$ jsou vektory. Pokud výsledek výpočtu determinantu vychází kladný, potom vektor *b* směřuje doprava od vektoru *a*, pokud je výsledek záporný, potom vektor *b* směřuje doleva od vektoru *a*, a pokud je determinant roven nule, potom mají vektory stejný směr. Následující algoritmus Alg. 1 v pseudokódu popisuje přesněji daný postup. Tento algoritmus lze nalézt v [angla97a].

```
Procedure FindAffectedArea(T:CDT, ab:Edge)
Begin
      t := Najdi trojúhelník, který obsahuje bod a je protnutý hranou ab;
      PU := Prázdný seznam;
      PL := Prázdný seznam;
      v := a;
      while (b neni v t) do
            tSEG := ProtějšíTrojúhelník(t, v);
            vSEG := ProtějšíBod(tSEG, t);
            if (vSEG je nad hranou ab)
            begin
                  Vloz(PU, vSEG);
                  v := bod sdílený t a tSEG nad hranou ab
            end
            else
            begin
                  Vloz(PL, vSEG);
                  v := bod sdílený t a tSEG pod hranou ab
            end;
            Odstraň t z T
            t := tSEG;
      end;
End;
```

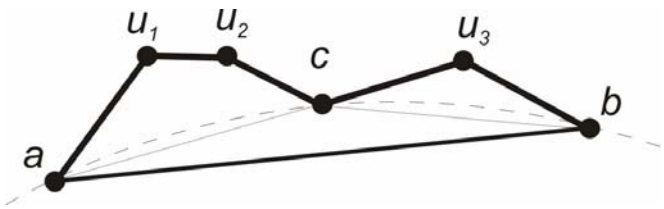
Alg. 1: Hledání zasažené oblasti

Triangulace zasažené oblasti

Takto vytvořená zasažená oblast má následující velmi užitečnou vlastnost. K vytvoření CDT stačí triangulizovat pouze tuto oblast. Důkaz je uveden v [angla97a]. Stručné vysvětlení: vyjmutím hran z triangulace a přidáním nových se nezmění konfigurace bodů a tak i nová Delaunayova triangulace zasažené oblasti bude splňovat podmínku prázdné opsané kružnice.

Triangulace zasažené oblasti se provádí zvlášť pro horní část a zvlášť pro část dolní. Toto je možné, protože obě oblasti na sebe navzájem nevidí přes vnucenou hranu. Dále je popsán způsob, jak algoritmus oblast triangulizuje.

Postup bude vysvětlen na triangulaci horní (levé) oblasti vytvořené vnucením hrany *ab*. Pro dolní (pravou) oblast je postup stejný. Je potřeba nalézt bod, jehož opsaná kružnice neobsahuje žádný další bod. Je to takový bod, který vytváří s body *a*, *b* největší opsanou kružnici. Pro tento bod nemusí platit, že je ze všech bodů nejblíže hraně *ab* i když to tak v dosti případech je. Tento bod je znázorněn na Obr. 3.3 jako bod *c*. Bod *c* se propojí s body vnucené hrany a vzniknou dvě nové hrany *ab* a *bc*. Tím vznikne i nový trojúhelník *abc*, jediný, který bude obsahovat vnucenou hranu. Zároveň vznikly dvě nové oblasti dané polygony tvořených pravou a levou částí horní hranice a nově vzniklými hranami. Na obrázku jsou to polygony $P1 = (a, u_1, u_2, c)$ a $P2 = (c, u_3, b)$. Pro každou nově vzniklou oblast rekurzivně opakujeme stejný postup jako pro celou zasaženou oblast. Rekurze končí ve chvíli, kdy nově vzniklá podoblast je trojúhelník.



Obr. 3.3: Triangulace horní části zasažené oblasti. Opsaná kružnice bodům a, b, c neobsahuje žádný jiný bod. Proto se oblast rozdělí na dvě části podle bodu c.

Výsledkem je tedy triangulace horní a dolní zasažené oblasti. Nově vzniklá triangulace je triangulace CDT. Takto může algoritmus vkládat další hrany do triangulace.

Následující kapitola se věnuje algoritmu S. W. Sloana.

3.2 Sloan

Algoritmus může být nalezen v [sloan93a]. Algoritmus používá lokální prohazování hran ke konstrukci CDT. Celý algoritmus je o něco složitější než postup, který zde uvedeme. Sloan mimo jiné popisuje datové struktury, které urychlují provádění metody. Zároveň je v článku popsán i způsob dělení oblasti na podoblasti, což urychluje výpočet. Omezíme se pouze na problémy, které se týkají CDT.

Sloan používá, stejně jako Anglada, zasaženou oblast stejně jako Anglada. Tuto oblast změní tak, aby se v ní vyskytla vnucená hrana, označme jí *ab*. Proces má dvě fáze. V první fázi se vnutí pomocí lokálního prohazování hran vnucená hrana *ab*. Tím vzniknou trojúhelníky, které ještě nejsou Delaunayovy. V další fázi se obě oblasti opraví tak, aby splňovaly Delaunayovo kritérium. K tomu se opět použije lokální prohazování hran.

Algoritmus v pseudokódu popisuje první část metody, vnucení hrany *ab*.

```
Najdi všechny hrany protnuté vnucenou hranou ab a ulož je do seznamu S
While (není seznam S prázdný) do
Begin
      e = Vyjmi hranu ze seznamu S.
      T1, T2 = Najdi trojúhelníky sdílející hranu e
      If  (T1 a T2 tvoří nekonvexní čtyřúhelník) /* prohození není možné */
            Zařaď e na konec seznamu S
      Else
      Begin
            Prohoď hranu e
            If e je stále protnutá hranou ab
                  zařaď e na konec seznamu S
      End
End /* while */
```

algoritmus 3-1: Vnucení hrany

Takto se tedy vnutí hrana *ab* do zasažené oblasti. Na počátku je seznam S vytvořen tak, že hrany jsou seřazeny podle toho, jak je algoritmus procházením od bodu *a* do bodu *b* nachází. Hledání protnutých trojúhelníků pracuje na principu podobném jako tomu bylo u algoritmu Anglada.

Poté, co je vnucena hrana, je potřeba opravit oblast tak, aby splňovala Delaunayovo kritérium. Toto se provádí pomocí prohazování hran. Hrany, které nesplňují Delaunayovo kritérium jsou prohazovány tak dlouho, dokud všechny hrany toto kritérium nesplňují. Jednoduše se prochází stále dokola všechny nové hrany. Pokud narazíme na hranu, která odpovídá vnucené hraně *ab*, přeskočíme ji. Tu prohodit nesmíme. Pokud již není žádná hrana, která by potřebovala prohodit, skončíme. Tímto je hrana vnucena a vzniká CDT.

Následující kapitola se věnuje algoritmu P. Maura.

3.3 Maur

Algoritmus P. Maura [maur04] využívá přechodu do vyšší dimenze a regulární triangulace. V této podkapitole bude rozebrán podrobně algoritmus i s teorií, která jej vysvětluje.

Mějme množinu bodů V v prostoru E^2 a množinu vstupních povinných hran E. Cílem je zkonstruovat CDT ze vstupních bodů obsahující zadané hrany E. Algoritmus nejprve vytvoří jakýmkoliv způsobem Delaunayovu triangulaci. Při konstrukci ignoruje vstupní hrany. Výsledkem jsou hrany triangulace E'.

Poté následuje hlavní část algoritmu, a totiž vnucení vstupních hran E. Algoritmus postupuje tak, že vnucuje hrany postupně, pro každou použije stejný postup. Nejprve je zapotřebí nalézt zasaženou oblast. K nalezení zasažené oblasti je možno použít stejný postup jako je použit v algoritmu M.V. Anglady[3]. K sestrojení CDT stačí triangulizovat pouze tuto zasaženou oblast pomocí Delaunayovy triangulace se zohledněním vnucené hrany. Toto je dokázáno v [angla97a]. Přesněji řečeno stačí zkonstruovat Delaunayovu triangulaci v horní a dolní zasažené oblasti[4].

V kapitole 2.3.3 je vysvětlen algoritmus přechodu do vyšší dimenze. Na tomto principu je založena konstrukce triangulace v zasažené oblasti. Základní myšlenka je zhruba následující:

---

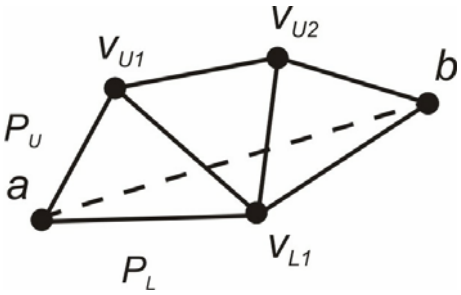
[3] Viz. kapitola 3.1, Anglada

[4] Horní a dolní zasažená oblast vysvětlena v kapitole 3.1, Anglada. Jedná se o seznamy v algoritmu označené P_U a P_L.

namapovat oblast na paraboloid tak, aby se vnucená hrana objevila v konvexní obálce sestrojené nad body v E^3. Jak přesněji namapování provést rozebírají následující odstavce.

Potřebujeme namapovat zasaženou oblast na paraboloid tak, aby při tvorbě konvexní obálky v ní vznikla jako konvexní hrana. Při použití jednoho paraboloidu je problém neřešitelný. Vnucená hrana, která neexistuje v Delaunayově triangulaci, způsobuje promítnutím na paraboloid jeho nekonvexitu. Tuto nekonvexitu není možné odstranit posunutím paraboloidu, protože konvexní obálka bodů vyzdvihnutých na paraboloid je nezávislá na posunutí středu paraboloidu. Řešením je použít dva paraboloidy a namapovat zvlášť horní a dolní oblast na vlastní paraboloid.

Mějme hranu *ab*, kterou chceme vložit do Delaunayovy triangulace (viz Obr. 3.4). Hrana *ab* rozdělí zasaženou oblast na horní a dolní oblast. Množiny bodů z horní a dolní zasažené oblasti jsou $V_U = (v_{U1}, v_{U2})$ a $V_L = (v_{L1})$. Množinu všech bodů, vzniklou spojením množin horní a dolní oblasti včetně bodů *a* a *b*, označme V^5.



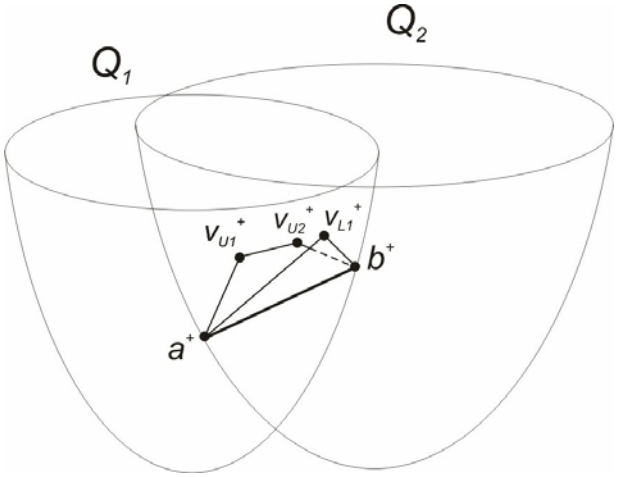
Obr. 3.4: Zasažená oblast při vkládání vnucené hrany *ab* do Delaunayovy triangulace. P_U a P_L jsou seznamy obsahující body horní a dolní oblasti.

Obě oblasti je potřeba namapovat na dva různé paraboloidy. Mějme dva různé paraboloidy Q_1 a Q_2. Rovnice paraboloidů zatím neznáme, budou odvozeny později. Nejprve je potřeba objasnit princip.

Všechny body z horní oblasti z množiny V_U namapujeme na paraboloid Q_1 způsobem, který je popsán v kapitole 2.3.3. Body z množiny V_L namapujeme na paraboloid Q_2. Budeme používat stejné značení jako v kapitole 2.3.3. Množinu bodů zasažené oblasti označme V. Množinu bodů vyzdvihnutých na paraboloid označme V^+. Její podmnožinu bodů vzniklých z bodů množiny V_U označme $V_U{}^+$. Podobně označme i množinu $V_L{}^+$ pro body dolní oblasti. Jakýkoliv bod v_i namapovaný na paraboloid označme $v_i{}^+$. Cílem je body namapovat tak, aby při vytvoření konvexní obálky tohoto tělesa jednak vznikla hrana a^+b^+, tak i se nemohly navzájem propojit body z množin P_U a P_L. Obě podmínky vyjadřují to samé. Tím, že v konvexní obálce bude hrana a^+b^+, nemůže existovat žádná další hrana protínající tuto hranu. Tím se zamezí jakémukoliv propojení bodů z protějších oblastí (například bodů v_{U1} a v_{L1}). Hrana a^+b^+ musí tedy být v prostoru E^3 umístěna tak, aby nezpůsobovala nekonvexitu.

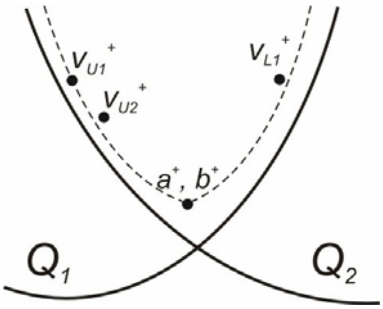
Paraboloidy nastavíme tak, aby existovaly průniky paraboloidů ve vrcholech vnucené hrany a^+ a b^+. Zároveň je potřeba, aby hrana a^+b^+ byla nejníže ze všech bodů. Díky tomu bude konvexita zaručena a hrana bude i v konvexní obálce. Situaci znázorňuje Obr. 3.5.

[5] Množina V obsahuje na Obr. 2.1 body *a, b,* V_{U1}, V_{U2}, V_{L1}.



Obr. 3.5: Zasažená oblast namapovaná na paraboloidy Q_1 a Q_2. Vnucená hrana je nejníže položená a proto body z protějších oblastí nemohou být přes ní propojeny hranou.

Z obrázku je vidět, jak mají paraboloidy vypadat. Horní a dolní oblast bodů se „nalepí“ na plochy dvou paraboloidů. Náhled na danou z boku situaci je na Obr. 3.6.



Obr. 3.6: Náhled z boku na body promítnuté na dvě plochy paraboloidů. Horní oblast je znázorněná vlevo, dolní vpravo. Body a^+ a b^+ v náhledu splynuly v jeden (b^+ je vzdálenější). Křivka vykreslená plnou čárou znázorňuje krajní oblast paraboloidu, čárkovaná křivka znázorňuje průnik dvou paraboloidů. Na nejnižším místě tohoto průniku leží body a^+, b^+.

Na tomto obrázku je vidět, že hrana a^+b^+ je nejníže položená ve směru osy Z. Body z protějších oblastí nevytvoří mezi sebou hrany až na body nejvýše položené. Jak je ale uvedeno v kapitole 2.3.3, promítají se zpět jen hrany z dolní části konvexní obálky. U příkladu zobrazeného na obrázku to bude hrana $v_{U1}{}^+v_{L1}{}^+$, která nebude zpětně promítnuta do triangulace. Jak je tedy na tomto příkladu vidět, všechny body se musí propojit přes vnucenou hranu a^+b^+.

Je ale zaručeno, že hrany konvexní obálky pro horní a dolní oblast vytvoří Delaunayovu triangulaci? Zaručeno to je a to tím, že při konstrukci konvexní obálky jedné oblasti jsou body druhé oblasti nepodstatné. To proto, že díky hraně a^+b^+ nikdy mezi sebou nemohou vytvořit hranu. Lze tedy teoreticky vytvořit konvexní obálku horní a dolní oblasti zvlášť a poté oblasti spojit. Dokonce některé algoritmy konstrukce konvexní obálky jsou na tomto principu založeny[6]. Jinak řečeno, konstrukce konvexní obálky se pro dvě oblasti stává lokálním problémem nezávislým na zbytku bodů.

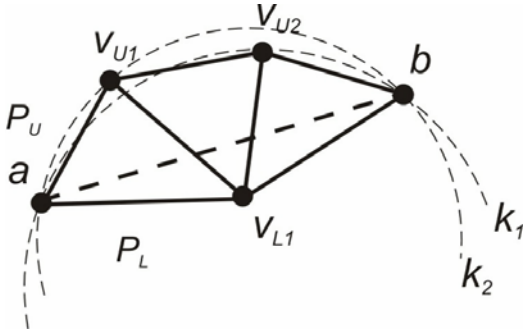
[6] Jsou to algoritmy typu rozděl a panuj.

Uvažujme triangulaci jedné oblasti bez jakékoliv znalosti bodů druhé oblasti. Zvolme například oblast horní s body z V_U. Body této oblasti leží na paraboloidu Q_2, který je zadán podle obecné rovnice paraboloidu. Není problém použít metodu přechodu do vyšší dimenze, jelikož jsou všechny podmínky splněny. Jak již bylo uvedeno, metoda na posunutí paraboloidu nezávisí. Stačí tedy pouze vytvořit konvexní obálku a promítnout spodní část obálky zpět do prostoru E^2. Tímto postupem získáváme Delaunayovu triangulaci jedné oblasti. Opakováním stejného postupu pro oblast druhou (dolní) získáváme druhou část Delaunayovy triangulace. Obě oblasti dohromady tvoří CDT, jak již bylo řečeno na začátku této podkapitoly.

Rovnice paraboloidů

Nyní se podíváme na to, jak budou definovány paraboloidy. Zaměřme se zatím pouze na hledání jednoho paraboloidu pro jednu z částí zasažených oblastí (například horní). Jediný požadavek je takový, aby hrana a^+b^+ ležela nejníže ze všech bodů obou oblastí. Na Obr. 3.7 je znázorněna hrana ab a dvě kružnice k_1 a k_2. Uvažme, že střed kružnice k_2 je i středem paraboloidu, který označme P. Pokud promítneme kružnici k_2 na paraboloid P, kružnice se promítne do prostoru jako kružnice k_2^+, která bude ve všech svých bodech mít stejnou výšku. To proto, že střed paraboloidu P je i střed kružnice k_2. Pokud bychom na tento paraboloid P promítli například kružnici k_1, kružnice by se nepromítla jako kružnice ale jako uzavřená křivka v prostoru E^3. Na paraboloid P promítneme hranu ab, získáváme a^+b^+. Body a^+a b^+ leží na kružnici k_2^+. Všechny body, které v prostoru E^2 leží uvnitř kružnice k_2 (například bod v_{U2}), se promítnou na body ležící níže než body kružnice k_2 a tím i níže než body a^+b^+. Toto je návodem, jak najít střed paraboloidu. Paraboloid musí mít takový střed, aby všechny ostatní vrcholy části oblasti ležely mimo kružnici k_2 (pro tento případ). Postup nalezení paraboloidu bude přesněji popsán v následujícím textu.

Pokud budou body ležet daleko od hrany a^+b^+, může mít paraboloid menší strmost stoupání v místě hrany. Pokud naopak budou body blízko hrany a^+b^+, musí paraboloid rychle stoupat, aby body vrcholy byly výš než body hrany. Paraboloid bude strmější v bodech vzdálenějších od středu. To znamená, že čím blíž budou body k vnucené hraně, tím bude muset mít paraboloid od hrany vzdálenější střed, aby v místě hrany byl strmější.



Obr. 3.7: Kružnice opsané zasažené oblasti. Kružnice k_1 protíná body a, b, v_{U1}. Kružnice k_2 prochází body a, b, v_{U2}. Kružnice k_2 neobsahuje žádné body, proto její střed bude i střed paraboloidu.

Obr. 3.7 znázorňuje zasaženou oblast s vnucenou hranou ab. Hledáme paraboloid takový, aby body a^+ a b^+ ležely ve stejné výšce vzhledem k souřadné ose Z[7]. Ve skutečnosti je možné,

[7] Situace je znázorněna na Obr. 3.6. Body a^+ a b^+ v náhledu splývají v jeden.

aby body vnucené hrany ve stejné výšce na ose *z* neležely. Byl však zvolen tento postup, protože je jednodušší na pochopení a implementaci.

Označme hledaný paraboloid *P* (je to buď paraboloid Q_1 pro horní oblast nebo Q_2 pro oblast dolní). Paraboloid *P* bude tedy mít tedy stejný střed jako kružnice opsaná bodům *a* a *b*. Tuto kružnici označme *k*. Takových kružnic opsaných je nekonečně mnoho. Další podmínkou je, aby hrana a^+b^+ ležela nejníže. To znamená, že všechny ostatní vrcholy příslušné části zasažené oblasti musí ležet mimo kružnici *k*. Proto hledáme takovou kružnici *k*, která neobsahuje ve svém vnitřku žádné body z části zasažené oblasti, pro kterou hledáme paraboloid. Jednoduchý a rychlý algoritmus nalezení středu kružnice *k* je popsán v následujícím odstavci.

Oblasti horní a dolní se prohledávají zvlášť. V jednom kroku pracujeme jenom s jednou oblastí (horní nebo dolní). Vstupem je seznam bodů části zasažené, označme jej **S**. Na začátku algoritmus uloží do proměnné **BOD** první bod v seznamu **S**. Tato proměnná uchovává třetí bod, který bude tvořit kružnici. Poté algoritmus cyklicky prochází všechny další body v seznamu kromě prvního, který byl pevně zvolen jako nejvhodnější bod na kružnici. Právě procházený bod je uložen v proměnné **Bi**. V následujících bězích cyklu jsou prováděny testy, zda kružnice tvořená body *a*, *b*, **BOD** obsahuje právě iterovaný bod **Bi**. Pokud ano, je třeba kružnici nově opsat bodům *a*, *b*, **Bi**. Bod **Bi** se v tom případě uloží do proměnné **BOD** a algoritmus pokračuje dále ve smyčce. Nová kružnice má větší poloměr, avšak pokrývá menší plochu z testované části zasažené oblasti. Nepokrývá žádnou plochu, která by nebyla pokryta kružnicí starou. Kružnice se sice změnila, ale není třeba testovat již otestované body, jelikož ty ležely mimo kružnici, která pokrývala dokonce větší plochu. Až algoritmus projde všechny body, získává kružnici s největším poloměrem tvořenou body *a*, *b* a **Bi**.

Ilustrace je na Obr. 3.7. Testujeme oblast horní, P_U. Nejprve vytvoříme kružnici obsahující body *a*, *b*, v_{U1} (na obrázku označená k_1). Při druhém kole cyklu testujeme bod v_{U2} a zjistíme, že bod leží uvnitř kružnice. Proto pro další cykly budeme již používat novou kružnici z bodů *a*, *b*, v_{U2} (na obrázku označenou k_2). Protože další žádné body v oblasti nejsou, algoritmus skončí a vrátí kružnici neobsahující žádný další bod, kružnici označenou k_2. Takto získáme kružnici jedním průchodem přes všechny body zasažené oblasti. Náročnost je tedy O(N), kde N je počet bodů zasažené oblasti. Pro urychlení není tvořena rovnice kružnice v průběhu iterací. Zjišťování parametrů kružnice je výpočetně náročné. V průběhu iterací stačí jen testovat, zda kružnice tvořená třemi body obsahuje daný testovaný bod. K tomu jsou používány determinantové testy z kapitoly 2.2, Rov. 3. Tyto testy jsou rychlejší než určování parametrů kružnice. Parametry kružnice potřebujeme znát až pro výslednou kružnici, abychom našli střed paraboloidu.

Když známe střed paraboloidu, potřebujeme ještě dopočítat jeho posunutí ve směru osy Z. Oba paraboloidy musí mít vlastnosti takové, že hranu a^+b^+ namapují na stejné místo v prostoru E^3. Řešení je snadné. Body hrany se dosadí do jednoho paraboloidu, kterému zvolíme parametr *c* náhodně[8]. Poté uděláme to samé pro druhý paraboloid. Rozdíl složky *z* pro bod a^+ namapovaný na první a druhý paraboloid připočteme k parametru *c* druhého paraboloidu.

[8] Parametr *c* určuje posunutí ve směru osy z. Rovnice je v kapitole 2.3.3, Rov. 2-4. Zvolit parametr můžeme například roven nule.

Vlastní konstrukce

V této podkapitole je vysvětleno, jak vlastní konstrukce CDT algoritmem Maur probíhá. Jedno řešení by mohlo být konstruovat konvexní obálku nad body namapovanými podle výše uvedeného postupu. Takovéto řešení problému je však numericky náročné. Existuje výpočetně rychlejší způsob, a to použití regulární triangulace. Jak již bylo řečeno v kapitole 2.6, konstrukci regulární triangulace je možné převést na konstrukci konvexní obálky bodů namapovaných na paraboloid posunutých o jejich váhu. Pokud váhy jsou nulové, body leží na paraboloidu a jejich regulární triangulace tvoří Delaunayovu triangulaci. Zde platí i obecnější pravidlo, že pokud body mají váhy takové, že výsledné pozice bodů leží na paraboloidu, jejich regulární triangulace tvoří také Delaunayovu triangulaci. Každý bod tedy může mít takovou váhu, aby jeho třetí souřadnice z odpovídala hodnotě požadované pro algoritmus (viz výše uvedený postup). Pro hodnotu třetí souřadnice z_i definujme rovnici:

$$v_i = x_i^2 + y_i^2 - w_i,$$

kde z_i je požadovaná výška i-tého bodu, x_i, y_i jsou souřadnice bodu a w_i je váha, kterou potřebujeme spočíst. Vztah $x_i^2 + y_i^2$ ve vzorci reprezentuje paraboloid se středem v počátku souřadné soustavy. Tento paraboloid je původní paraboloid, na který jsou implicitně mapovány všechny body při použití regulární triangulace (poté je od nich odečtena váha). Cílem je tedy najít takové váhy w_i, aby přesunuly body z tohoto paraboloidu na dva výše odvozené paraboloidy. Vzorec jednoduše upravíme na:

$$w_i = x_i^2 + y_i^2 - z_i.$$

Takto vypočtenou váhu w_i použijeme jako váhu do výpočtu regulární triangulace. Horní část zasažené oblasti mapujeme na paraboloid Q_1, dolní část na paraboloid Q_2. Mapování provedeme tím, že do rovnice paraboloidu dosadíme mapovaný bod. Výsledná výška je hodnota z, z které dopočítáme váhu w. Pro body a a b jsou hodnoty v stejné při mapování na oba paraboloidy. Váhy w vycházejí však odlišné pro paraboloid Q_1 a paraboloid Q_2. Přitom nezáleží na tom, na který paraboloid bod mapujeme, protože paraboloidy jsou nastaveny tak, aby se protínaly v bodech a^+ a b^+.

Poté sestrojíme regulární triangulaci. To provedeme pomocí zásobníkové metody. Na počátku zásobník obsahuje všechny hrany. V cyklu vyjímáme hrany ze zásobníku, dokud není zásobník prázdný. Každou hranu, která není lokálně regulární, prohodíme. Zároveň do zásobníku zařadíme všechny vnější hrany, pokud tam ještě nejsou. Pokud narazíme na lokálně regulární hranu, jen ji vyjmeme ze zásobníku a nic dalšího s ní neprovádíme. Algoritmus je podobný algoritmu lokálnímu prohazování hran z kapitoly 2.3.1. Po vyprázdnění zásobníku zůstávají všechny hrany lokálně regulární. Jak již bylo zmíněno výše, hrany v tomto stavu tvoří CDT.

Pokud vnucujeme vzájemně blízké hrany, algoritmus, jak je popsaný výše, by mohl zrušit předchozí povinnou hranu. To proto, že hrana by mohla být uvnitř zasažené oblasti nové hrany. Proto je třeba do algoritmu prohazování hran přidat podmínku, aby algoritmus nikdy neprohazoval dříve zařazenou povinnou hranu. Poté metoda funguje bez problémů.

Metoda se ukázala být funkční na většinu svých vstupů. V některých případech selhala. Důvodem byly numerické chyby. Jejich rozbor, návrh jejich odstranění a další popis výsledků je v kapitole 8.3.

4 Komprese

Jedním z cílů diplomové práce je vyzkoušet algoritmus Maur jako kompresní algoritmus trojúhelníkových sítí. Této problematice se věnuje tato kapitola.

4.1 Existující metody

Tato kapitola se věnuje kompresím triangulací pomocí Delaunayovy triangulace. Reálná data se mohou skládat z několika stovek tisíců bodů. Přenos takového množství dat po síti může být časově náročné. Proto je komprese trojúhelníkových sítí často probíraným tématem.

Stručně zde budeme prezentovat metodu komprese trojúhelníkových sítí za použití Delaunayovy triangulace. Tato metoda může být nalezena v [kim99a]. Metoda je založena na poznatku, že většina trojúhelníkových sítí je podobná sítím Delaunayovým. To znamená, že většina hran těchto sítí je Delaunayových. Stačí tedy uložit jen body a rozdíl mezi hranami v Delaunayově triangulaci a původní triangulaci. Není potřeba ukládat Delaunayovy hrany, protože tyto hrany mohou být vytvořeny konstrukcí Delaunayovy triangulace.

Rozdíl hran je uložen následovně. Triangulaci, kterou chceme komprimovat, označme *T*, její vrcholy označme *V*. Mějme dva seznamy hran. Jeden seznam ukládá hrany, které v triangulaci chybí, aby triangulace byla Delaunayova. Tento seznam označme *M*[9]. Hrany, které jsou v komprimované triangulaci navíc oproti Delaunayově triangulaci, označme *P*[10]. Při kódování sestrojíme nad body z množiny *V* Delaunayovu triangulaci, označme ji *DT*. Každou hranu z *DT*, která není v *T*, uložíme do seznamu *M*. Každou hranu z *T*, která není v *DT*, uložíme do seznamu *P*. Hrany, které jsou jak v *T* tak i v *DT*, neukládáme, protože budou vytvořeny pomocí Delaunayovy triangulace. Příklad je na Obr. 4.1.

Metoda navíc používá vylepšení na ukládání hran, které tvoří v grafu strom. Výsledky, které autoři článku naměřili, jsou příznivé. Algoritmus je schopen zakódovat hrany s kompresním poměrem 0.23 bitů na vrchol.

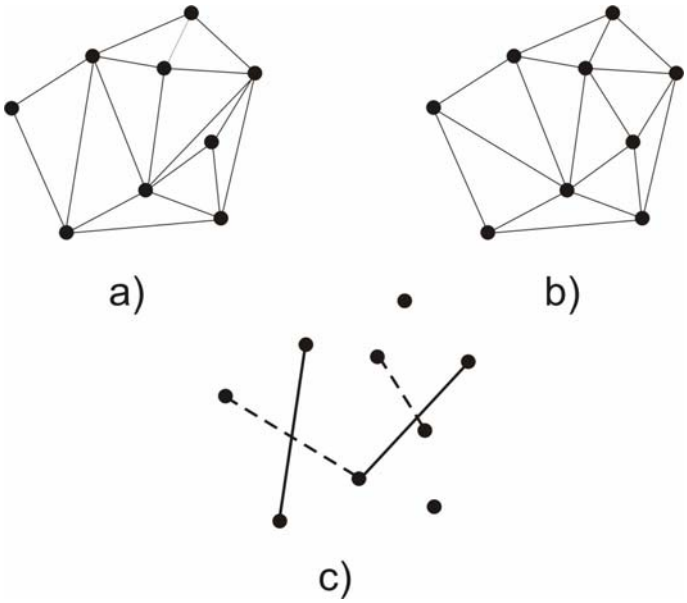
4.2 Komprese pomocí algoritmu Maur

Jedním z cílů diplomové práce je vyzkoušet použití algoritmu Maur na komprese triangulací. Problém je podobný jako v předchozí v podkapitole 4.1. Potřebujeme nějak efektivně zakódovat hrany. Princip této metody je použití CDT. Triangulaci, kterou chceme komprimovat označme *T,* množinu jejích vrcholů označme *V*. Delaunayovu triangulaci vytvořenou nad množinou bodů *V* označme *DT*. Dále se budeme zabývat tím, jak zakódovat hrany, které se vyskytují v *T*, ale nejsou v *DT*. Zkusme uvážit postup, kde každou takovou hranu vnutíme jako povinnou hranu za použití algoritmu Maur.

---

[9] Označení *M* je zkratka pro *Mínus*. Seznam obsahuje hrany, které se při dekompresi od Delaunayovy triangulace musí odečíst, aby vznikla původní triangulace.

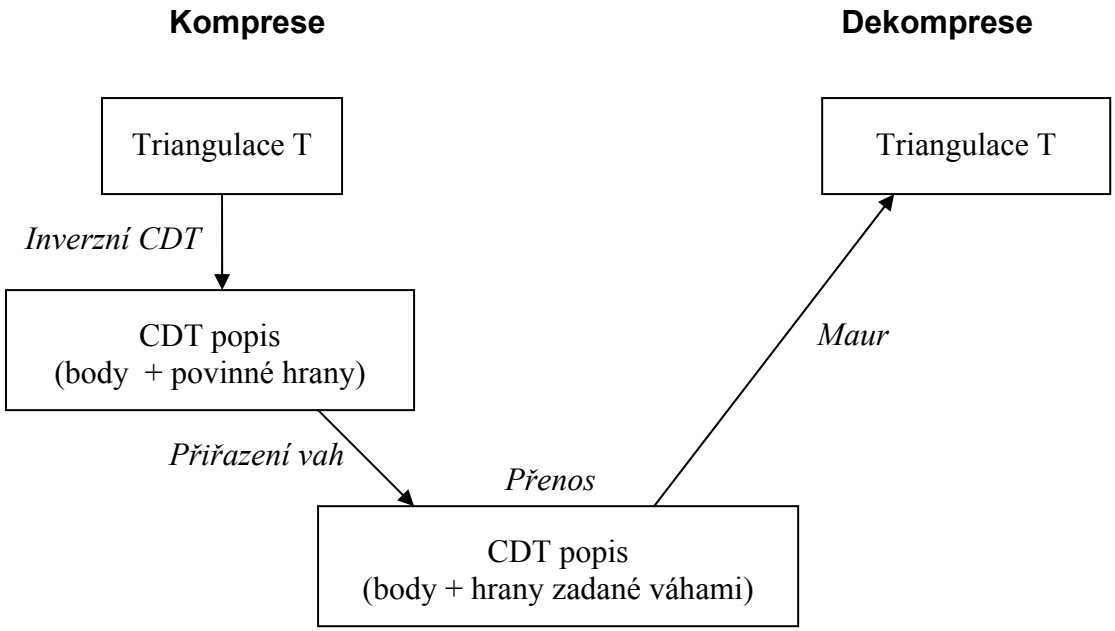
[10] Označení P je zkratka pro Plus. Seznam obsahuje hrany, které se při dekompresi musí přičíst k Delaunayově triangulace, aby vznikla původní triangulace.



Obr. 4.1: Komprese pomocí Delaunayovy triangulace. Obrázek a) znázorňuje komprimovanou triangulaci. Obrázek b) znázorňuje Delaunayovu triangulaci vytvořenou nad stejnou množinou bodů. Obrázek c) zobrazuje rozdíl mezi a) a b). Čárkované jsou čáry, které jsou v b) navíc, plnou čárou jsou znázorněny hrany, které v b) chybí.

Cílem je vytvořit kompresi, kde se ukládají body a jejich váhy. Dekomprese je založena na tom, že předává body a jejich váhy algoritmu Maur, který nejprve vytvoří normální Delaunayovu triangulaci a potom pomocí vah bodů vnutí potřebné hrany. Co tedy potřebujeme ke vnucení jedné hrany: je zapotřebí předat váhy bodů v zasažené oblasti vnucené hrany. Algoritmus Maur poté vnutí požadovanou hranu. Problém je však ten, že při tomto vnucování hrany změní i hrany jiné. Počáteční úvaha by se dala upravit tak, že nehledáme rozdíl triangulace *T* od triangulace *DT*, ale hledáme popis původní triangulace *T* jako CDT. Schéma komprese je na Obr. 4.2. Algoritmus komprese by tedy měl za úkol najít CDT popis k dané triangulaci T. Tento postup zatím nebyl nikde v této práci popsán. Předpokládejme, že jsme schopni k dané triangulaci najít její popis jako CDT. To znamená, že dokážeme libovolnou triangulaci popsat jako množinu bodů a množinu vnucených hran tak, aby konstrukce CDT nad touto množinou vytvořila původní triangulaci T. Dále tento CDT popis upravíme tak, že pro každou hranu vypočteme váhy bodů, jak to je popsáno v algoritmu Maur. To znamená, že namísto hran uložíme pouze váhy bodů v zasažené oblasti. Toto je výsledný komprimovaný formát.

Dekomprese by potom vypadala tak, že by se uložený vstup použil pro algoritmus Maur. Přesněji pouze pro jeho část vnucení hrany pomocí regulární triangulace. Váhy bodů jsou totiž již přepočítány. Výsledkem by bylo CDT, které odpovídá původní triangulaci T.



Obr. 4.2: Schéma a komprese

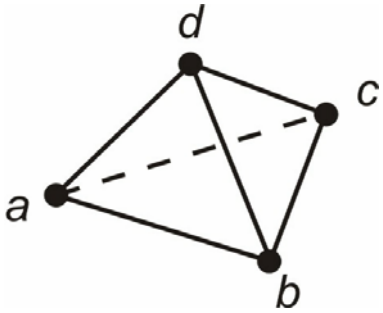
Při kompresi každou vnucenou hranu popíšeme pomocí vah bodů v její zasažené oblasti. Zde však vyvstává velká nevýhoda tohoto algoritmu. Dostáváme pro každou hranu alespoň čtyři hodnoty vah. Čtyři proto, že zasažená oblast se skládá ze dvou bodů vnucené a nejméně dvou dalších bodů uzavírající polygon. Ve většině případů je však bodů v zasažené oblasti více. Váhy regulární triangulace jsou reálná čísla. Je ale možnost při implementaci ušetřit místo použitím jiného datového typu, například integeru. To proto, že algoritmus výpočtu nezávisí na přesných hodnotách, ale na vzájemném poměru velikosti vah bodů. Jednoduše řečeno, stačí pro stejnou výslednou konfiguraci hran jen to, že váha je například dvakrát větší než nějaká jiná váha a už nezáleží na přesných hodnotách vah. Proto je možné nepoužívat reálné datové typy. Každá hrana však přesto potřebuje na zakódování alespoň čtyři celá čísla navázaná nějakým způsobem na příslušné body. Toto navázání však není úplně jednoduché. Pokud máme dvě vnucené hrany *e1* a *e2*, které jsou v těsné blízkosti, jejich zasažené oblasti se protínají. Potom i body těchto zasažených oblastí potřebují dvoje váhy, jedny pro hranu *e1* a druhé pro hranu *e2*. Při ukládání vah je tedy třeba u vah rozlišit, ke které hraně se váží. To přináší další data, která je potřeba uchovávat. Oproti tomu pouhé zadání hrany pomocí dvou indexů bodů, které hrana propojuje, je paměťově méně náročné. Ušetříme nejméně dvě celá čísla na uložení jedné hrany. Vnucení hrany pomocí zadání vah okolních bodů je tedy neefektivní. Původní záměr algoritmu byl uložit ke každému bodu váhu a potom sestrojit pomocí algoritmus Maur CDT. Kvůli výše uvedenému problému je takový přístup neefektivní.

Dalším problémem by při použití takového algoritmu mohlo být hledání popisu CDT ke vstupnímu grafu. Jedná se o problém provádění inverzního algoritmus k algoritmu konstrukce CDT. Hledání takové metody by překračovalo rozsah diplomové práce, proto zde žádný postup popsán nebude.

Závěrem lze říci následující. Došli jsme k závěru k tomu, že použití algoritmu Maur pro komprese sítí nepřináší vylepšení kompresního poměru oproti jiným snazším metodám. Kompresní poměr by byl vyšší oproti běžnému uložení trojúhelníkové sítě. Lepšího poměru by se dosáhlo pouze, pokud by síť byla velmi podobná Delaunayově triangulaci, jelikož bychom nemuseli kódovat mnoho hran. I tak je ale lepší použít jednodušší metodu rozdílu hran popsanou výše.

5 Delaunayova Triangulace v E^3

Tato kapitola se věnuje Delaunayově triangulaci v E^3. Delaunayova triangulace v E^3 propojuje body do tetrahedronu (čtyřstěnu), viz Obr. 5.1.



Obr. 5.1: Tetrahedron *a*, *b*, *c*, *d*.

Stejně jako trojúhelník, tak i tetrahedron má orientaci. Na obrázku je tetrahedron definovaný pořadím vrcholů *a*, *b*, *c*, *d*. Tuto orientaci nazveme pravotočivou. Opačnou orientaci by měl tetrahedron definovaný *a*, *b*, *d*, *c*. Tuto orientaci nazveme levotočivou. Každý tetrahedron sousedí se čtyřmi tetrahedrony stejně jako trojúhelník sousedí se třemi trojúhelníky. Delaunayova triangulace v prostoru E^3 je také nazývaná tetrahedronizace.

5.1 Definice a pojmy

Nyní definujme triangulaci v E^3.

Definice 5-1, Triangulace v E^3:

Nechť V je množina vrcholů v prostoru E^3. Triangulace T je množina čtyřstěnů, pro něž platí následující vlastnosti:

1. bod $p \in E^3$ je vrcholem čtyřstěnu tehdy a jen tehdy, pokud p patří do množiny V.
2. průsečík dvou čtyřstěnů z množiny T je buď prázdný nebo se jedná o společnou stěnu, hranu a nebo vrchol. [11]

Definice má stejný význam jako definice pro Delaunayovu triangulaci. Jediný rozdíl je, že je rozšířena o jednu dimenzi. Pro E^3 triangulaci platí stejná pravidla jako pro E^2 triangulaci. Následující definice popisuje Delaunayovu triangulaci v E^3.

Definice 5-2, Delaunayova triangulace v E^3:

Nechť V je množina vrcholů v prostoru E^3. DT je triangulace nad množinou bodů V, pro které platí, že koule opsaná každému čtyřstěnu neobsahuje žádný další bod z V.

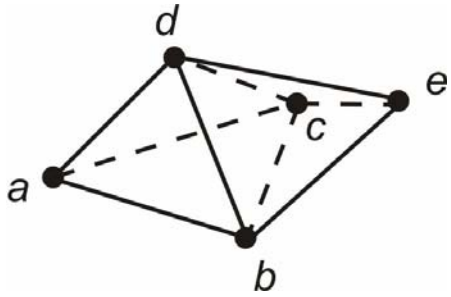
Tato definice říká to samé, jako definice pro Delaunayovu triangulaci v E^2. Jediný rozdíl je v tom, že testy se provádí pomocí koule opsané čtyřstěnu.

Potřebné pojmy pro triangulaci v E^3 jsou protější tetrahedron, protější bod a soused přes trojúhelník.

[11] Definice byla upravena a převzata z [koho02a].

Definice 5-3, Protější tetrahedron vzhledem k bodu:

Mějme tetrahedron *a*, *b*, *c*, *d* a sousední tetrahedron *c*, *b*, *e*, *d*, jak je znázorněno na Obr. 5.2. Protější tetrahedron k tetrahedronu *a*, *b*, *c*, *d* vzhledem k bodu *a* je sousední tetrahedron *c*, *b*, *e*, *d.*



Obr. 5.2: Tetrahedron *a, b, c, d* se sousedním tetrahedron *c, b, e, d.*

Definice 5-4, Protější bod vzhledem k bodu a tetrahedronu:

Mějme tetrahedron *a*, *b*, *c*, *d* a sousední tetrahedron *c*, *b*, *e*, *d*, jak je znázorněno na Obr. 5.2. Protější bod k tetrahedronu *a*, *b*, *c*, *d* a bodu *a* je bod *e*.

Definice 5-5, Soused přes trojúhelník:

Mějme tetrahedron *a*, *b*, *c*, *d* a sousední tetrahedron *c*, *b*, *e*, *d*, jak je znázorněno na Obr. 5.2. Soused k tetrahedronu *a*, *b*, *c*, *d* přes trojúhelník *b*, *c*, *d* je tetrahedron *a*, *b*, *c*, *d*.

5.2 Testování Delaunayova kriteria

Při testování Delaunayova kriteria postupujeme stejně jako v E^2 variantě. Následující definice definuje pojem Delaunayova lokálně optimálního trojúhelníku.

Definice 2-5, Delaunayův lokálně optimální trojúhelník:

Nechť *b*, *c*, *d* je trojúhelník společný dvěma tetrahedronů *a*, *b*, *c*, *d* a *c*, *b*, *e*, *d*. Situaci znázorňuje Obr. 5.2. Trojúhelník *b*, *c*, *d* je lokálně optimální podle Delaunayova kritéria, pokud koule opsaná tetrahedronu *a*, *b*, *c*, *d* neobsahuje bod *e*.

Pokud je kriterium splněno tak, jak je definováno, je vždy platné i pro druhý tetrahedron. To znamená, že koule opsaná tetrahedronu *c*, *b*, *e*, *d* neobsahuje bod *a*. Kriterium prázdné opsané koule lze testovat podobným způsobem jako kriterium prázdné opsané kružnice. Implicitní rovnice koule je

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2 = 0,$$

Rov. 8: Implicitní rovnice koule

kde $[x_0, y_0, z_0]$ je střed koule, $[x, y, z]$ je dosazovaný bod, který chceme testovat. Pokud je rovnice splněna, potom dosazovaný bod leží na povrchu koule. Pokud výraz na levé straně rovnice vychází menší než nula, bod leží uvnitř koule, pokud tento výraz vyjde větší než nula, bod leží vně koule. I zde vyvstává stejný problém s nalezením parametrů koule. Existuje výpočet pomocí determinantu, který je odvozen z rovnice Rov. 8. Test lze provést bez znalostí parametrů koule. Výpočet se provádí dosazením bodů do determinantu 5x5:

$$\det = \begin{vmatrix} a_x & a_y & a_z & a_x^2 + a_y^2 + a_z^2 & 1 \\ b_x & b_y & b_z & b_x^2 + b_y^2 + b_z^2 & 1 \\ c_x & c_y & c_z & c_x^2 + c_y^2 + c_z^2 & 1 \\ d_x & d_y & d_z & d_x^2 + d_y^2 + d_z^2 & 1 \\ e_x & e_y & e_z & e_x^2 + e_y^2 + e_z^2 & 1 \end{vmatrix}$$

Rov. 9: Test bodu uvnitř koule

, kde $a = [a_x, a_y, a_z]$, $b = [b_x, b_y, b_z]$, $c = [c_x, c_y, c_z]$ a $d = [d_x, d_y, d_z]$ jsou body tvořící tetrahedron, kterému je opsána koule. Bod $e = [e_x, e_y, e_z]$ je testovaný bod. Výsledek determinantu určuje pozici testovaného bodu e vzhledem ke kouli opsané bodům a, b, c a d. Pokud determinant vychází větší než nula, leží bod e uvnitř koule. Pokud determinant vychází menší než nula, leží bod e vně koule. Pokud je determinant roven nule, bod e leží na povrchu koule. Stejně jako determinant testu bodu uvnitř kružnice je možné zjednodušit[12], podobnou úpravu lze provést i zde. Determinant lze redukovat na determinant 4x4:

$$\det = \begin{vmatrix} a_x - e_x & a_y - e_y & a_z - e_z & (a_x - e_x)^2 + (a_y - e_y)^2 + (a_z - e_z)^2 \\ b_x - e_x & b_y - e_y & b_z - e_z & (b_x - e_x)^2 + (b_y - e_y)^2 + (b_z - e_z)^2 \\ c_x - e_x & c_y - e_y & c_z - e_z & (c_x - e_x)^2 + (c_y - e_y)^2 + (c_z - e_z)^2 \\ d_x - e_x & d_y - e_y & d_z - e_z & (d_x - e_x)^2 + (d_y - e_y)^2 + (d_z - e_z)^2 \end{vmatrix}.$$

Rov. 10: Test bodu uvnitř koule po úpravě

Dosazením bodů do determinantu a jeho vypočtením lze jednoduše zjistit, zda bod leží uvnitř, vně anebo na kouli. Výpočet je závislý také na orientaci bodů. Znaménkové testy výše popsané platí pro tetrahedrony orientované levotočivě. Pokud je tetrahedron orientovaný pravotočivě, výsledný determinant je potřeba před porovnáním znaménka vynásobit hodnotou *-1*.

Pokud bod leží uvnitř koule, je zapotřebí pozměnit triangulaci tak, aby bod ležel vně koule. Zde již není problém tak přímočarý jako je to v E^2 triangulaci. V E^2 triangulaci stačí jednoduše prohodit hranu. Zde je prohození komplikovanější. V E^2 dva trojúhelníky tvořící dohromady nekonvexní čtyřstěn jsou vždy vzhledem k sobě navzájem Delaunayovy. V E^3 případě to tak být nemusí. Proto je třeba prohazovat i dva nekonvexní tetrahedrony. Dokonce existují konfigurace, které prohodit nejdou. Typům prohození se věnuje následující kapitola.

5.3 Prohození konfigurace tetrahedronů

V E^2 triangulaci je prohození jednoduchou záležitostí. Každé dva trojúhelníky mají dvě možné konfigurace. Pokud tvoří konvexní čtyřúhelník, pokaždé bylo možné konfiguraci prohodit (viz. kapitola 2.2). Pokud měly trojúhelníky nekonvexní tvar, jejich sdílená hrana sice prohodit nejde, ale není to potřeba. To proto, že dva nekonvexní trojúhelníky vždy splňují Delaunayovo kritérium.

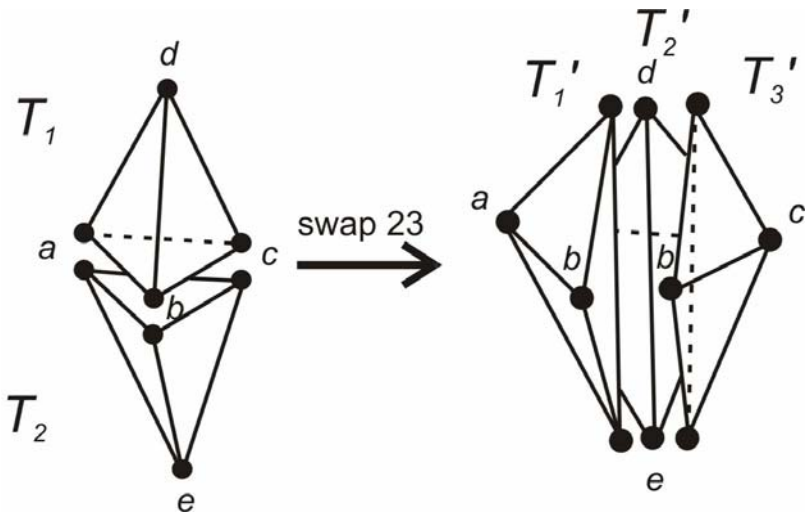
U Delaunayovy triangulace v E^3 je prohození konfigurace dvou tetrahedronů komplikovanější. Za prvé mohou existovat tetrahedrony, které prohodit nejdou, za druhé

[12] Test bodu uvnitř kružnice lze nalézt v kapitole 2.2, Delaunayova triangulace, rovnice Rov. 2: Test bodu uvnitř kružnice upravená na vztah Rov. 3: Test bodu uvnitř kružnice po úpravě.

existuje i více možností prohození. Postupně zde popíšeme a vysvětlíme různé varianty prohození konfigurace dvou a více tetrahedronů. Zároveň budou vysvětleny i případy, kdy prohození není možné. Prohození se dost často označuje anglickým slovem *swap*.

Swap 23

Swap 23 je swap, při kterém jsou dva tetrahedrony rozloženy na tři (proto 2→3). Tento typ prohození konfigurace znázorňuje Obr. 5.3.

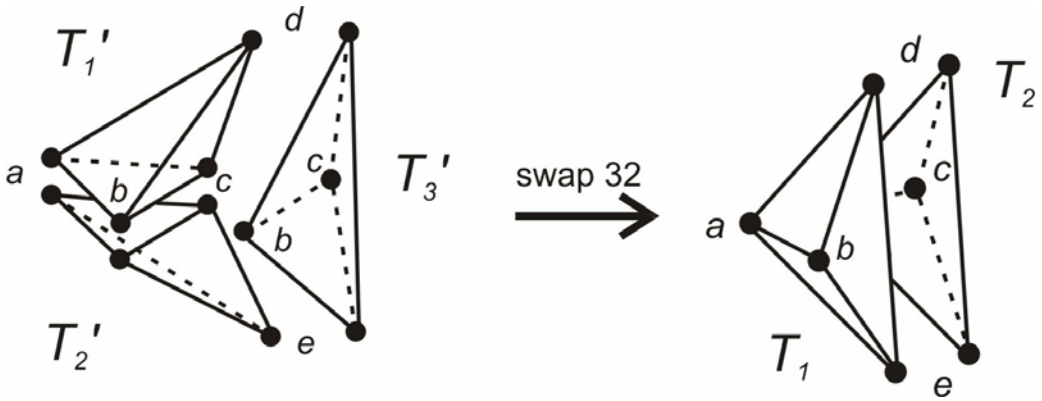


Obr. 5.3: Swap 23. Dva sousední tetrahedrony T_1, T_2 jsou při swapu 23 rozloženy na tři tetrahedrony T_1', T_2' a T_3'. Tetrahedrony jsou pro názornost vykresleny odděleně.

Swap 23 je možný provést jen se dvěma tetrahedrony v **konvexním** tvaru. Dva tetrahedrony mají konvexní tvar, pokud diagonála mezi dvěma jejich navzájem nesdílenými body (*ed*) protíná sdílený trojúhelník (*abc*). Konvexnost tetrahedronů je jedinou podmínkou provedení swapu 23. Pokud je tato podmínka splněna, prohození je vždy možné provést.

Swap 32

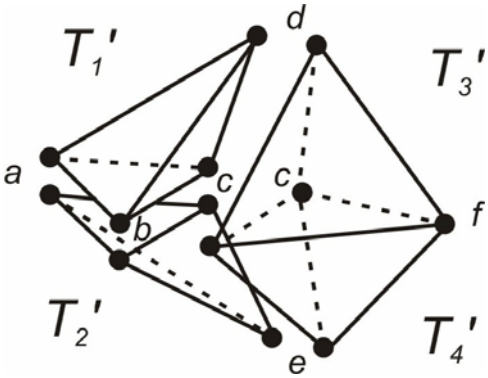
Swap 32 je prohození konfigurace, při kterém tři tetrahedrony se spojí ve dva. Situace je znázorněna na Obr. 5.3. Jedná se o opačný swap ke swapu 23. Tetrahedrony T_1', T_2' a T_3' se spojí v tetrahedrony T_1 a T_2. Další příklad je na Obr. 5.4. Zde je situace přehlednější.



Obr. 5.4: Swap 32. Tetrahedrony T_1', T_2' a T_3' se spojily v tetrahedrony T_1 a T_2. Pravý obrázek je mírně pootočen okolo svislé osy, aby byl vidět i bod *c*.

Podmínkou pro tento swap je **nekonvexnost** dvou tetrahedronů T_1' a T_2'. Swap 32 sice používá tři tetrahedrony, ale swap je proveden právě kvůli nesplnění podmínky prázdné opsané koule dvou tetrahedronů. Pokud tedy zjistíme, že dva tetrahedrony jsou nekonvexní, je

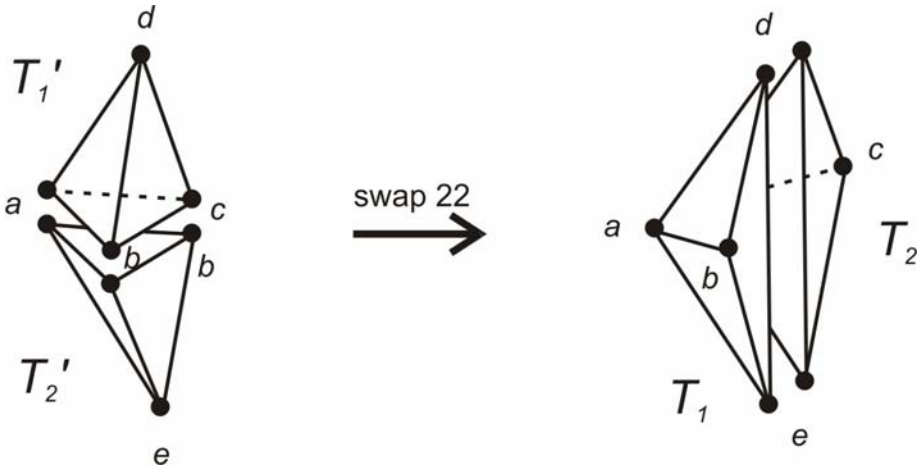
třeba najít v síti třetí tetrahedron. Na Obr. 5.4 třetí tetrahedron existuje (je označen T_3'). Ne pokaždé však v síti je třetí tetrahedron vhodný k provedení swapu. Dva tetrahedrony T_1' a T_2' na Obr. 5.5 nemohou provést swap 32, protože v síti není třetí tetrahedron. Na místo toho jsou tam dva tetrahedrony T_3' a T_4'. Swap by byl možný, pokud by se nejprve prohodily tetrahedrony T_3' a T_4' swapem 23.



Obr. 5.5: Dva nekonvexní tetrahedrony T_1' a T_2' nemohou provést swap 32, protože jim k tomu chybí třetí tetrahedron. Místo třetího tetrahedronu jsou v síti dva tetrahedrony T_3'a T_4'.

Swap 22

Zatím jsme vysvětili dva typy swapů. Swap 23 prohazoval dva konvexní tetrahedrony, swap 32 prohazoval dva nekonvexní tetrahedrony. Další zatím neuvedenou možností jsou dva tetrahedrony, jejichž dva trojúhelníky leží v jedné rovině. Tyto tetrahedrony jsou znázorněny na Obr. 5.6.

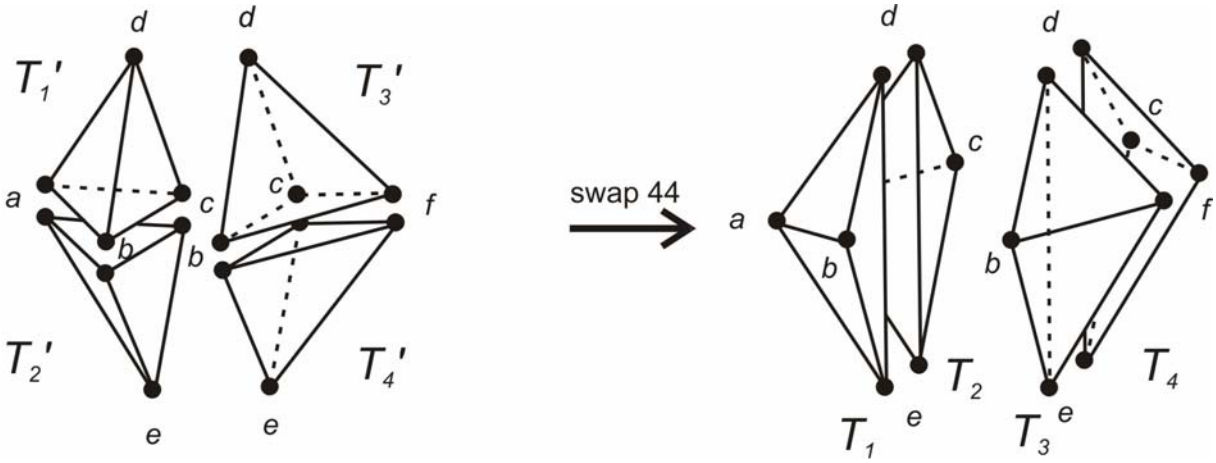


Obr. 5.6: Swap 22. Prohazují se dva tetrahedrony T_1' a T_2'. Jejich body b, c, d a e leží v rovině. Výsledkem jsou dva tetrahedrony T_1 a T_2, kde opět body b, c, d a e leží v rovině.

Prohozením těchto tetrahedronů vzniknou dva nové tetrahedrony, jejichž trojúhelníky opět leží v rovině. Podmínka pro swap 22 je, že jejich dva trojúhelníky musejí ležet v rovině a zároveň tyto trojúhelníky musejí být na okraji sítě, to znamená, že trojúhelníky jsou součástí konvexní obálky. Pokud by totiž napravo od tetrahedronů T_1' a T_2' byly další dva tetrahedrony, nemohli bychom prohodit jen tetrahedrony T_1' a T_2'. V tom případě bychom museli použít swap 44.

Swap 44

Swap 44 se používá v případech jako swap 22 jen s rozdílem, že sousedi přes trojúhelníky ležící v rovině existují. To znamená, že tetrahedrony neleží na kraji konvexní obálky. Jedná se vlastně o dvakrát provedené prohození. Swap 44 demonstruje Obr. 5.7. Provedeme swap 22 na tetrahedrony T_1' a T_2' (vznikají T_1 a T_2) a zároveň provedeme swap na tetrahedrony T_3' a T_4' (vznikají T_3 a T_4).



Obr. 5.7: Swap 44. Trojúhelníky b, c, d a b, e, c dvojic tetrahedronů T_1', T_2', T_3' a T_4' leží v rovině. Po prohození vznikají tetrahedrony T_1, T_2, T_3 a T_4.

Podmínky pro provedení swapu 44 jsou následující. Dva tetrahedrony musí ležet v rovině a sousedi přes rovinné trojúhelníky musí obsahovat stejnou konfiguraci tetrahedronů na druhé straně.

5.4 Algoritmy DT v prostoru E^3

Jako algoritmus konstrukce Delaunayovy triangulace v prostoru E^3 zde rozebereme algoritmus inkrementální konstrukce. Postup je podobný postupu popsanému v kapitole 2.3.2.

Inkrementální vkládání

Tento algoritmus je online algoritmem. Přesto na počátku potřebujeme znát rozsah vstupní množiny bodů. Princip je podobný jako v E^2 přístupu, i zde je potřeba nejprve sestavit hlavní tetrahedron, do kterého budou vkládány vstupní body. Tento tetrahedron, označme T_0, body tohoto čtyřstěny budou na konci algoritmu z triangulace odstraněny. Tetrahedron T_0 musí být tak velký, aby všechny vstupní body ležely uvnitř tohoto tetrahedronu. Teoreticky by bylo dobré, kdyby jeho body ležely v nekonečnu. Při realizaci jsou však hodně vzdálené body nebezpečné kvůli numerickým problémům.

Do tetrahedronu T_0 se vkládají postupně vstupní body. Při vložení prvního bodu se rozdělí tetrahedron T_0 na 4 menší tetrahedrony. V tomto kroku nic dalšího neprovádíme. Při vložení dalších bodů je potřeba vždy najít tetrahedron, v kterém se nachází nově vložený bod. Tento tetrahedron je potřeba rozdělit. Může vzniknout různý počet tetrahedronů v závislosti na tom, zda bod vložíme do prostoru tetrahedronu, na hranu a nebo na stěnu tetrahedronu. Po vzniku nových tetrahedronů je potřeba všechny vnější trojúhelníky, které náleží novým tetrahedronům, lokálně optimalizovat podle Delaunayova kritéria. Vnější trojúhelníky jsou všechny trojúhelníky náležící novým tetrahedronům kromě navzájem sdílených trojúhelníků. Pokud vnější trojúhelníky nejsou lokálně optimální, provedeme na příslušně tetrahedrony sdílející trojúhelník vhodný swap. swapu vzniknou nové tetrahedrony, jejich množinu označme N. Po tomto swapu máme zaručeno, že trojúhelníky sdílené těmito tetrahedrony

budou lokálně optimální. Testovat se dále budou jen trojúhelníky, které jsou vzhledem k ostatním tetrahedronům z množiny N vnější. To znamená, že testujeme trojúhelníky navzájem nesdílené tetrahedrony z množiny N. Pokud nejsou lokálně optimální, je potřeba je prohodit a rekurzivně opakovat stejný postup na všechny vnější trojúhelníky.

Prohození je vždy možné jenom jedno ze čtyř možností. Zde je rozdíl oproti E^2 variantě, a to, že prohození nemusí vždy být možné provést. Takový případ nastane, pokud prohazujeme dva tetrahedrony mající dohromady nekonvexní tvar. Jedná se o případ popsaný v kapitole 5.3, kde ke swapu 32 chybí třetí tetrahedron. Tento lze řešit pomocí fronty. Pokud prohození není možné, vloží se daný trojúhelník do fronty a prohození se provede až to možné bude. Do fronty se vkládají všechny trojúhelníky, které nejsou lokálně optimální. Oblast je obnovena ve chvíli, kdy se fronta vyprázdní. Druhá možnost je algoritmus implementovat pomocí rekurze. Pokud narazíme na trojúhelník, který nelze prohodit, dále se o něj nestaráme. Algoritmus totiž tento trojúhelník optimalizuje později sám. To znamená, že se rekurzivní postup na trojúhelník vždy dostane i jinou cestou. Varianta s frontou a varianta s rekurzí vždy konverguje.

Pokud jsou vloženy všechny body, je zapotřebí odstranit z triangulace čtyři body uměle vytvořené pro tetrahedron T_0. Po odstranění bodů a k nim patřících krajních tetrahedronů, získáváme Delaunayovu triangulaci v E^3 pro vstupní množinu bodů.

6 CDT E^3

V této kapitole bude popsán problém CDT pro E^3. Zároveň budou popsány algoritmy, pro jejichž implementaci jsme se rozhodli.

Nejprve definujme viditelnost. Definice je pouhým rozšířením definice viditelnosti z kapitoly 2.5 o jednu dimenzi.

Definice 6-1, Viditelnost v E^3:

> Dva vrcholy *a* a *b* v E^3 triangulaci *T* jsou navzájem viditelné, pokud úsečku *ab* neprotíná žádný trojúhelník z triangulace *T*.

Definice popisuje viditelnost tak, jak je intuitivně chápána. Body na sebe vidí, pokud jim v tom nebrání žádná stěna nějakého tetrahedronu. Další definice popisuje CDT E^3.

Definice 6-1, CDT E^3:

> Nechť *V* je množina bodů v prostoru E^3. Mějme množinu trojúhelníků, které jsou tvořeny nad body z množiny *V* a navzájem se neprotínají. Tuto množinu označme *F*. CDT je množina tetrahedronů tvořených jednak z trojúhelníků F jednak i z přidaných trojúhelníků *F'*. Vytvořené tetrahedrony musí splňovat následující podmínky:
>
> - Tetrahedrony tvoří triangulaci. To znamená, že splňují všechny podmínky dané definicí 5-1.
> - Koule opsané tetrahedronům nesmějí obsahovat žádný vrchol triangulace viditelný z vnitřních bodů tetrahedronu. Viditelnost se v tomto případě vztahuje pouze na trojúhelníky *F*.[13]

Definice popisuje CDT pro E^3 tak, jak je tomu v případě pro E^2. Rozdíl je zde pouze v rozšíření o jednu dimenzi. Při testování Delaunayova kritéria uvažujeme jenom body, které jsou viditelné z vnitřku tetrahedronu vzhledem ke hranám *F*. To znamená, že ve viditelnosti nám nebrání hrany *F'*.

Použití CDT je stejné jako v E^2 případě. Nejčastější využití je obnovení nekonvexního tvaru hranice nějakého tělesa.

6.1 Algoritmy CDT v prostoru E^3

Pro konstrukci CDT E^3 neexistuje mnoho algoritmů. Budou zde popsány dva algoritmy: zametací algoritmus a algoritmus inkrementálního vkládání hran pomocí swapů. Oba dva algoritmy pochází od Jonathana Richarda Shewchuka[14].

6.1.1 Inkrementální algoritmus

Popis algoritmu může být nalezen v [shew03a]. Inkrementální algoritmus je inkrementálním z hlediska postupného vkládání povinných hran do triangulace. Toto je rozdíl od konstrukce Delaunayovy triangulace, kde inkrementální algoritmus byl ten, který postupně vkládá body.

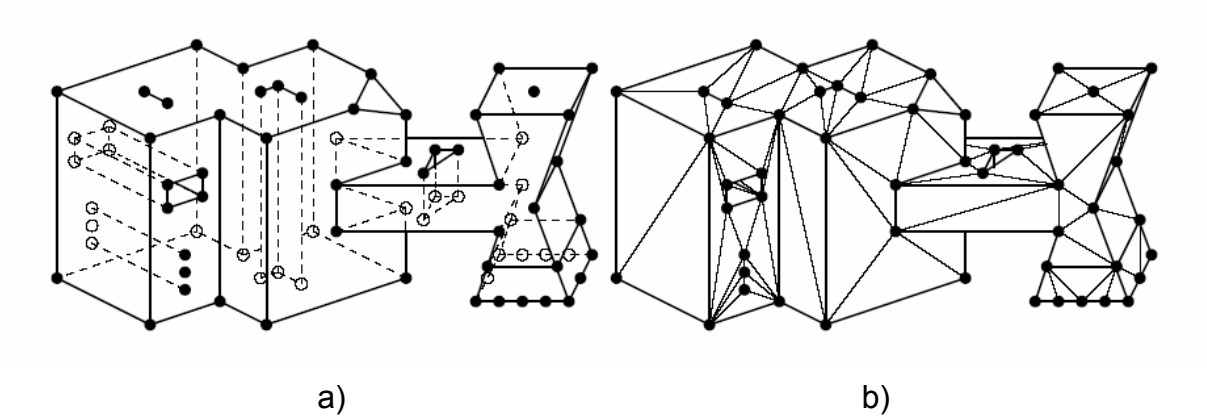
---

[13] Definice byla převzata z [koh02a].

[14] Jonathan Richard Shewchuk působí na Kalifornské univerzitě v Berkeley. Jeho oborem jsou převážně meshe a jejich konstukce. V článcích zabývajících se problematikou CDT je často citován (ne jenom v nich).

Tento algoritmus je pro náš výzkum metod v E^3 poměrně důležitý, proto bude podrobněji popsán.

Algoritmus je v článku popsán obecně pro N dimenzí a funguje i pro regulární triangulaci. Algoritmus tvoří CDT z takzvaného PLC (Piecewise Linear Complex). PLC je znázorněno na Obr. 6.1 a). Na obrázku b) je výsledek algoritmu, CDT vytvořené z PLC.

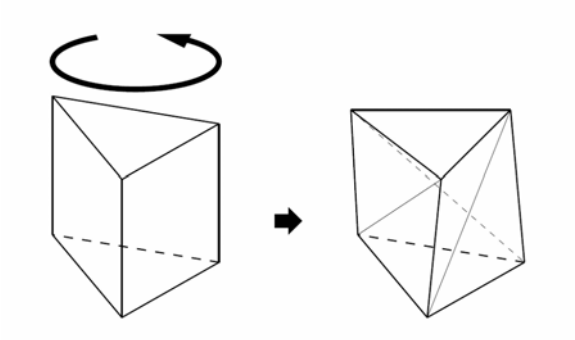


Obr. 6.1: Piecewise linear complex (PLC) na a) a jeho CDT na obrázku b). Obrázek je převzatý z [shew02a].

Jak je vidět na Obr. 6.1, PLC je celkem komplikovaný útvar. Může obsahovat díry, samostatné body a vnitřní řezy. Ve skutečnosti PLC je množina bodů, povinných trojúhelníků a povinných hran. Povinnými hranami se pro prostor E^3 v této práci nezabýváme. Algoritmus je použitelný jen pro určitá PLC. Jde o to, že ne pro všechny PLC existuje CDT. V článku jsou popsány podmínky pro existenci CDT k danému PLC. Tyto podmínky jsou definovány pro libovolný počet dimenzí. Zde si uvedeme pouze variantu, která odpovídá konstrukci CDT v prostoru E^2 a E^3. Ke každému trojrozměrnému PLC existuje CDT, pokud každá jeho hrana splňuje podmínku, že existuje koule opsaná této hraně, která neobsahuje žádný jiný bod uvnitř koule *ani na povrchu* této koule. Pro náš případ to znamená, že pokud dokážeme ke každé hraně povinného trojúhelníku najít takovou opsanou kouli, která nebude obsahovat žádný další bod, CDT je možné zkonstruovat. Toto je však postačující podmínka, ne nutná. To znamená, že i když taková koule neexistuje, je možné že CDT bude přesto existovat, jen to není stoprocentně zaručeno. Pro případ dvojrozměrný je podmínka definována pro prázdné opsané kružnice každého bodu. Toto je vždy splněno. Proto pro E^2 existuje vždy CDT. Více detailů o tom, kdy CDT existuje může být nalezeno v [shew02a].

Pokud pro PLC neexistuje CDT, lze upravit PLC pomocí přidání bodů tak, že CDT bude existovat. Bod je do PLC přidán poměrně složitými metodami popsanými v [shew02a]. Podrobněji se přidáním bodu zabývat nebudeme.

Existuje i případ, kdy neexistuje *žádná* triangulace zadané oblasti. Tímto případem je například Schönhardtův polyhedron znázorněný na Obr. 6.2. Ten obsahuje tři diagonální hrany, které jsou povinné. Neexistuje způsob, jak tento polyhedron jakkoliv triangulizovat. Pokud se však do vnitřku tělesa přidá vhodně jeden bod, triangulace je možná.



Obr. 6.2: Schönhardtův polyhedron je vytvořen otočením jednoho konce tělesa vlevo. Tím vznikají tři diagonální hrany.

Pro následující text se omezme pouze na vysvětlování problematiky konstrukce CDT. Předpokládejme tedy, že pro dané PLC existuje CDT. PLC převedeme na množinu bodů a povinných trojúhelníků. Zároveň se omezíme pouze na vkládání jednoho trojúhelníku do triangulace. Označme V vstupní množinu bodů, F množinu povinných trojúhelníků a t aktuálně vkládaný trojúhelník.

Vstupem pro algoritmus je zkonstruovaná Delaunayova triangulace nad množinou bodů. Algoritmus vnucení trojúhelníku je podobný algoritmu Maur z kapitoly 3.3. Používá také přechodu do vyšší dimenze a regulární triangulace. Přestože algoritmus zvládá tvořit i CDT pro regulární triangulaci, vysvětlení zaměříme pouze na konstrukci CDT pro Delaunayovu triangulaci.

Algoritmus nejprve nalezne zasaženou oblast. Nalezení zasažené oblasti je podobné dvojdimenzionálnímu případu a bude podrobně rozebráno v kapitole 6.2. Každému bodu horní a dolní části zasažené oblasti přiřadí váhu w_i rovnou 0. Algoritmus uchovává svůj vnitřně počítaný čas r. Princip je založen na postupném zvyšování tohoto svého času r. Váha každého bodu w_i se mění v závislosti na čase r. Vztah pro výpočet váhy bodu x_i pro aktuální čas je

$$w_i = r * d(t, x_i)$$

Rov. 11: Váhy bodů

, kde t je povinný trojúhelník, $d(t, x_i)$ je vzdálenost bodu x_i od trojúhelníku t a r je čas. Takto máme přiřazené váhy k bodům v závislosti na čase. Na základě těchto vah bude prováděna regulární triangulace zasažené oblasti. Na počátku jsou váhy nulové, to znamená, že všechny trojúhelníky v oblasti jsou lokálně regulární[15]. Důvod je ten, že pokud jsou váhy všech bodů nulové, jedná se o speciální případ, kterým je regulární triangulace.

Algoritmus používá prioritní frontu, do které na začátku zařadí všechny trojúhelníky, které jsou protnuté vnucený trojúhelníkem t. Určení priority, se kterou jsou trojúhelníky do fronty zařazovány, zde nebude přesně popisováno. Prioritou trojúhelníku je čas, kdy přestane být daný trojúhelník lokálně regulární. Trojúhelníky jsou z fronty vybírány podle právě aktuálního času. Vybraný trojúhelník je prohozen pomocí vhodného swapu. Určení vhodného swapu bylo popsáno v kapitole 5.3. Po prohození se stává trojúhelník lokálně regulární. Do fronty zařadíme vnější trojúhelníky tetrahedronů, které vznikly po provedení swapu. Při zařazení do fronty je trojúhelníku opět přiřazena priorita podle výše popsaného postupu.

[15] Lokálně regulární trojúhelník sice v diplomové práci nebyl definován, ale byla definována lokálně regulární hrana (viz. kapitola 2.6). Lokálně regulární trojúhelník je pouhým rozšíření definice do E^3.

Algoritmus takto zvyšuje čas a provádí swapy trojúhelníků dokud se fronta nevyprázdní. Po vyprázdnění fronty je trojúhelník *t* do oblasti vnucen a oblast splňuje kritéria CDT.

Algoritmus používá ještě tzv. Pertubation method, která zamezuje singulárním případům, které by mohly vzniknout v určitém čase *r*. Tyto případy by vážně mohly narušit chod algoritmu, proto je její použití velmi důležité. Metoda však není tolik důležitá pro tuto práci, proto bude její popis vynechán.

Princip algoritmu by se dal vysvětlit následujícím způsobem. Jeho princip je podobný principu algoritmu Maur z kapitoly 3.3. Využívá také přechodu do vyšší dimenze a regulární triangulace. U algoritmu Maur se tvoří dva paraboloidy, na které se zasažená oblast promítne tak, že povinná hrana je součástí konvexní obálky. Představme si výše popsaný algoritmus aplikovaný na triangulaci v prostoru E^2. Zde se hrana promítá na jeden paraboloid, který se postupně zakřivuje tak, aby se hrana vyskytla v konvexní obálce. Zakřivení paraboloidu je způsobeno váhami w_i, které jsou přiřazeny bodům zasažené oblasti. Tyto váhy se zvětšují v závislosti na čase (viz rovnice Rov. 11). Váhy jsou závislé i na vzdálenosti od povinné hrany (resp. povinného trojúhelníku pro E^3). Čím vzdálenější je bod od hrany, tím rychleji roste jeho váha v závislosti na čase. Váhy bodů vnucené hrany se nemění, protože vzdálenost je nulová. Tím se horní a dolní oblasti postupně zvyšují, dokud se vnucená hrana neobjeví v dolní části konvexní obálky.

Podobnost s algoritmem Maur je v principu konstrukce. Algoritmus je však navržen pro všechny dimenze a zvládá i regulární triangulaci. Pozvolné měnění vah zamezuje možnému zaseknutí algoritmu. Algoritmy využívající swapy ke vnucení trojúhelníku jsou náchylné na zaseknutí algoritmu. V [joe89a] je dokázáno, že není vždy možné změnit libovolnou triangulaci na cílovou triangulaci pomocí sekvence swapů. V některých případech může dojít k zaseknutí. Dokonce ne každý postup swapů k cíli vede a to i přesto, že oblast je možné pomocí swapů na cílovou přeměnit. Proto používá algoritmus systém prioritní fronty, která určuje správné pořadí swapů.

Algoritmus obsahuje i další části, které nebudou popsány. Metoda je poměrně složitá na implementaci.

6.1.2 Zametací algoritmus pro konstrukci CDT ve vyšších dimenzích

Popis algoritmu může být nalezen v [shew00a]. Algoritmus pracuje na principu zametacích algoritmů. Tyto algoritmy nejsou inkrementální, to znamená, že potřebují znát vstupní množinu před začátkem vykonávání konstrukce. Algoritmus tvoří zároveň nepovinné hrany i hrany povinné. Zde je rozdíl například oproti algoritmu Maur, Anglada nebo Sloan, které vkládají hrany do triangulace postupně. Do hotových CDT umí tyto algoritmy stále přidávat nové hrany. Toto neplatí u zametacího algoritmu.

Tento algoritmus má za vstup body a povinné trojúhelníky mezi těmito body. Algoritmus má dvě varianty. Triangulace je tvořena buď *zametáním* rovinou nebo koulí. Varianta s rovinou pracuje tak, že se pohybuje rovina kolmá na jednu zvolenou osu ve směru této osy. Pro jednoduchost zvolme například osu *X* a rovinu *ZY*. Tato rovina je na počátku v záporné ose *X*. Vše, co je od roviny ve směru osy *X*, pojmenujme tak, že to leží *před* rovinou. Vše ležící v záporném směru osy *X* je *za* rovinou. Určení *před* a *za* odpovídají tomu, jak se rovina pohybuje. Na počátku jsou tedy všechny body před rovinou. Rovina se pohybuje vpřed ve směru osy *X*. Jak postupně prochází vstupními body, konstruuje triangulaci. Vše za rovinou je hotová triangulace, vše před rovinou je zatím nezpracovaný vstup.

To, jak se triangulace konstruuje nebudeme vzhledem k náročnosti postupu rozebírat. Pro tuto práci není tento algoritmus příliš důležitý.

Hrubý popis konstrukce je následující. Postupná konstrukce probíhá tak, že k nově vzniklým nedokončeným stěnám tetrahedronům se postupně připojují nové body, které rovina protíná. Pokud je tetrahedron jednou dokončen, nebude již nikdy změněn. Vždy vznikají nové nedokončené stěny těsně za rovinou. Body, které se mohou propojit se stěnou trojúhelníků se určují pomocí testů prázdné opsané koule. Při tomto postupu jsou respektovány povinné hrany.

Varianta zametání koulí je podobná zametání rovinou. Namísto roviny se postupně uvnitř vstupní množiny zvětšuje koule. Vše, čím stěna koule projde, se konstruuje podobným principem, jak bylo uvedeno u roviny. Když je koule tak velká, že celý vstup leží uvnitř, je triangulace hotová. Varianta s koulí je lepší pro vstupy, kde není známá hranice triangulace. Oproti variantě s rovinou je však náročnější na implementaci a používá komplikovanější matematiku.

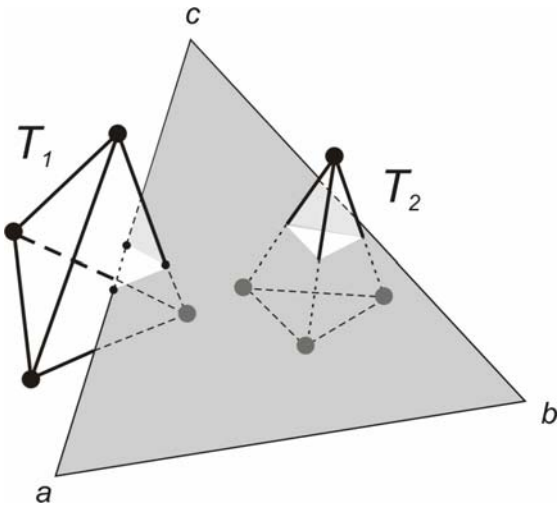
Složitost algoritmu je $O(N^2)$. V praxi je průměrný čas na konstrukci sítě $O(N^2 + N \log N)$. Nejrychlejší výsledky jsou dosaženy při konstrukci sítě tvořené z vnucených trojúhelníků organizovaných do tvaru hvězdy. V takovém případě je složitost $O(N \log N)$.

Implementace obou algoritmů je poměrně složitá. Jejich naprogramováni by nebylo v rámci diplomové práce možné. Přesto je možné využít znalostí těchto algoritmů pro konstrukci vlastních metod. Inkrementální algoritmus bude využíván při vysvětlování v dalších částech diplomové práce. Dalším úkolem diplomové práce je vyzkoušet rozšíření metody Maur do prostoru E^3. Nejprve proto bude rozebráno hledání zasažené oblasti, které používají všechny níže popsané algoritmy.

6.2 Hledání zasažené oblasti

Všechny uvedené algoritmy používají zasaženou oblast, ve které potom mění konfiguraci tetrahedronů tak, aby se v ní vyskytl vnucený trojúhelník. V případě pro prostor E^2 byla zasažená oblast nalezena tak, že se zařadily do seznamu všechny trojúhelníky, které byly protnuté vnucenou hranou. V prostoru E^3 je problém analogický. Zasažená oblast se skládá ze všech tetrahedronů, které jsou protnuté vnuceným trojúhelníkem. Vnucený trojúhelník označme *t*. Tetrahedron je protnutý vnuceným trojúhelníkem *t* tehdy a jen tehdy, pokud nějaký jeho trojúhelník je vnuceným trojúhelníkem *t* protnutý. To znamená, že se do zasažené oblasti patří i hraniční tetrahedrony, které jsou protnuté třeba jen z části. Tetrahedron protnutý a tetrahedron protnutý jen z části, je zobrazen na Obr. 6.3.

V prostoru E^3 není situace tak snadná, jako je to v prostoru E^2. V následujícím textu budou popsány dvě metody hledání zasažené oblasti, metoda ohraničení oblasti a metoda brutální síly. Obě tyto metody používají hledání průsečíků. Proto nejprve bude popsán mechanismus hledání průsečíků.



Obr. 6.3: Ukázka protnutých tetrahedronů. Tetrahedron T_2 je protnutý celý, tetrahedron T_1 je protnutý jen z části.

Hledání průsečíků

Dost často potřebujeme hledat průsečíky trojúhelníku s přímkou nebo průsečík trojúhelníku s trojúhelníkem. Jedna možnost je použít projekci ve směru přímky na jednu z rovin XY, YZ nebo XZ. Tímto způsobem hledáme průsečík trojúhelníku *t* a přímky *p*. Promítání ve směru přímky je ekvivalentní kosoúhlému promítání známému z počítačového grafiky[16]. Tím se promítne trojúhelník z trojrozměrného prostoru do dvojrozměrného. Jelikož promítáme ve směru přímky *p*, přímka *p* se promítne do roviny jako bod. Tím se převede trojrozměrný problém na dvojrozměrný. Vzájemná poloha bodu a trojúhelníku se dá řešit s pomocí orientačních testů popsaných v kapitole 3.1. Pro náš problém je navíc důležitá pouze informace, zda bod leží uvnitř trojúhelníku, vně anebo na hranách trojúhelníku.

Je třeba určit rovinu, na kterou se bude promítat. Ta se určí tak, že spočteme skalární součin normalizovaného vektoru přímky s normálou každé ze tří rovin XY, YZ nebo XZ. Pro projekci bude vybrána ta z těchto tří rovin, pro níž skalární součin vychází největší. Skalární součin určuje, nakolik dva vektory směřují stejným směrem. Normála roviny je kolmice na rovinu. Vybíráme tedy rovinu takovou, aby úhel mezi vektorem přímky *p* a normálou zvolené roviny byl co nejbližší úhlu kolmému.

Metoda je náchylná na numerické chyby. Nepříjemné případy nastávají, pokud jsou na sebe vektor přímky s normálou trojúhelníku téměř kolmé. Je třeba testovat tyto speciální případy, aby nedošlo k dělení nulou. Problém spočívá v tom, že o takových závažných věcech se algoritmus musí rozhodnout ještě před tím, než spočte výsledek.

Tento problém odstraňuje použití Plűckerových souřadnic.

Plűckerovy souřadnice

Tyto souřadnice se používají k určování vzájemných poloh přímek v prostoru. Přímka v Plűckerových souřadnicích je reprezentována pomocí 6 reálných hodnot. Mějme přímku zadanou dvěma body, kterými přímka prochází, body $a=(a_x, a_y, a_z)$ a $b=(b_x, b_y, q_z)$. Plűckerovy souřadnice, označme je $L_0, L_1, .. , L_6$, se vypočtou podle následujícího předpisu[17]:

[16] Anglický termín pro kosoúhlé promítání je *oblique projection*.

[17] Předpis může být nalezen v [plucker].

$$L_0 = a_x \; b_y - b_x \; a_y$$
$$L_1 = a_x \; b_z - b_x \; a_z$$
$$L_2 = a_x - b_x$$
$$L_3 = a_y \; b_z - b_y \; a_z$$
$$L_4 = a_z - b_z$$
$$L_5 = b_y - a_y$$

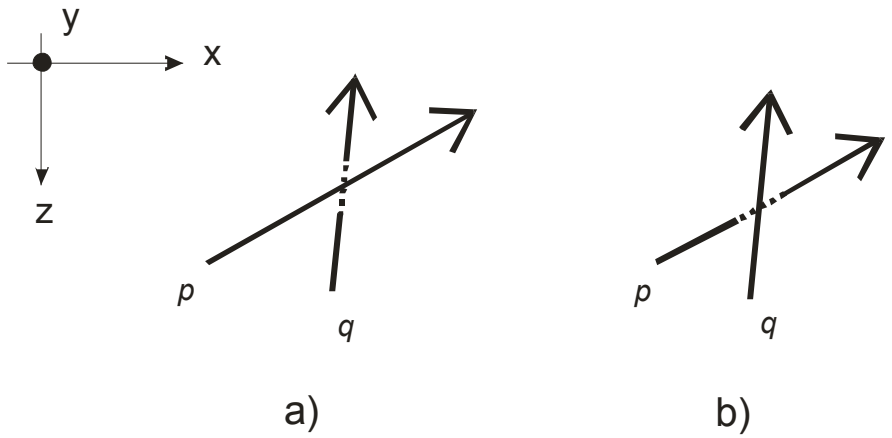
(Rov. 12: Výpočet Plűckerových souřadnic)

Při pohledu na výpočet Plűckerových souřadnic je zřejmé, že by výpočet neměl přinášet žádné numerické problémy, protože vzorce neobsahují dělení. Pro Plűckerovy souřadnice je definován operátor, který nad nimi pracuje. Tento operátor se nazývá side operátor a z jeho výsledku lze určit vzájemnou polohu přímek. Side operátor je definován následovně:

$$side(P, Q) = P_0 Q_4 + P_1 Q_5 + P_2 Q_3 + P_3 Q_2 + P_4 Q_0 + P_5 Q_1,$$

(Rov. 13: Side operátor)

kde P, Q jsou Plűckerovy souřadnice dvou libovolných přímek p a q. Výsledek side operátoru je reálné číslo. Pokud je výsledek roven nule, přímky se buď protínají nebo jsou rovnoběžné. Pokud je výsledek rozdílný od nuly, znaménko výsledku určuje vzájemnou polohu přímek. Dvě možnosti jsou znázorněny na Obr. 6.4.



Obr. 6.4: Vzájemná poloha dvou přímek. Na obrázku a) je přímka *p* nad přímkou *q*, na obrázku b) je přímka *p* pod přímkou *q*. Osa y směřuje směrem k pozorovateli. Výsledek *side(P, Q)* je menší než 0 pro a), větší než 0 pro b).

Obrázek znázorňuje dvě možnosti konfigurace přímek. Pro výsledek je důležité to, jaký mají přímky mezi sebou úhel. Pokud směřují stejným směrem (skalární součin vektorů přímek je větší než 0), je výsledek jiný, že když jedna přímka směřuje opačným směrem, než přímka druhá.

Jednoduše by se dal side operátor přímky p s přímkou q popsat následovně. Představme si, že promítneme dvě přímky v prostoru do roviny tak, aby přímka p byla blíže k pozorovateli než přímka q. Tato situace je znázorněna na Obr. 6.4 a). V bodě, kde se v náhledu přímky protínají, je přímka p výše než přímka q ve směru osy Y. Znaménko *side(p, q)* potom určuje, zda přímka q směřuje od přímky p doprava nebo doleva. Pokud přímky leží v rovině, potom side operátor vychází roven nule. Toto nastane, pokud jsou přímky rovnoběžné, nebo se protínají.

Ze znalosti polohy dvou přímek lze vyvodit postup zjištění, zda je trojúhelník protnutý přímkou. Mějme přímku *p* a trojúhelník *abc*. Pro každou jeho hranu *ab*, *bc*, *ca* spočteme side operátor s přímkou *p*. Získáváme tři reálné hodnoty $s_1 = side(ab, p)$, $s_2 = side(bc, p)$ a $s_3 = side(ca, p)$. Pokud všechny hodnoty jsou větší než nula, nebo všechny hodnoty jsou menší než nula, potom je trojúhelník protnutý přímkou. Tento fakt zde odvozen není, vyplývá přímo z popisu významu side operátoru.

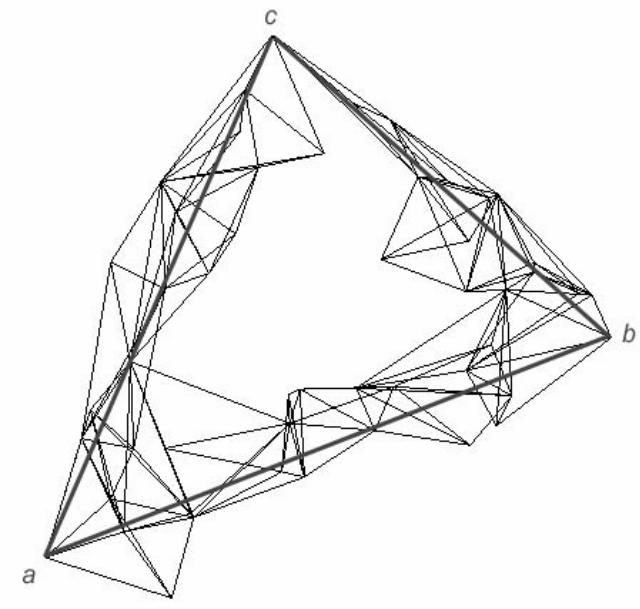
Průsečík trojúhelníku s trojúhelníkem se spočte tak, že se testují všechny hrany jednoho trojúhelníku vůči trojúhelníku druhému a naopak. Při testování hran se však takto převádí hrana na přímku. To způsobí, že postup výše popsaný bude indikovat průsečík, i když průsečík neexistuje. Je to tím, že se ve skutečnosti se testuje přímka namísto hrany. Metodu je třeba doplnit o orientační testy. Body testované hrany musí být každý opačným směrem od trojúhelníku, aby se jednalo o průsečík.

Metoda hledání průsečíků pomocí Plückerových souřadnic je numericky stabilní, jelikož nepoužívá dělení. Zároveň je její implementace snadná.

Metoda ohraničení oblasti

Jedna z metod hledání zasažené oblasti je metoda vyplnění ohraničené oblasti. Princip metody spočívá v tom, že se najdou tetrahedrony protnuté hranami vnuceného trojúhelníku. Uvažujme vnucený trojúhelník *abc*, označme jej *t*. Hledání těchto tetrahedronů probíhá pomocí procházení po sousedech protnutých hranou. Hledání začíná v bodě *a* a postupuje směrem k bodu *b*. Nejprve je potřeba najít tetrahedron obsahující bod *a*, který je protnutý úsečkou *ab*. Poprvé je to snadné, pokračujeme dále do tetrahedronu sdíleného přes trojúhelník protější k bodu *a*. Dostaneme se do dalšího tetrahedronu. V tomto tetrahedronu nalezneme ten trojúhelník, který je protnutý hranou *ab*. Takové trojúhelníky jsou dva. Jeden ten, kterým jsme do tetrahedronu přišli a druhý ten, kterým odejdeme do dalšího tetrahedronu. Takto pokračujeme, dokud se nedostaneme do bodu *b*.

Tento postup opakujeme i pro hranu *bc* a hranu *ca*. Výsledkem je hranice zasažené oblasti. Ukázka zasažené oblasti je na Obr. 6.5. Seznam tetrahedronů hranice označme *H*.



Obr. 6.5: Hranice zasažené oblasti. Tlustou čarou jsou znázorněny hrany trojúhelníku.

Dalším krokem je vyplnění vnitřku oblasti. Jedna možná metoda je následující. Vytvoříme seznam tetrahedronů, označme *S*. Do seznamu zařadíme ty tetrahedrony, které jsou protnuté rovinou tvořenou trojúhelníkem *abc*. Nalezne se jakýkoliv tetrahedron ze seznamu *S*, který leží mimo oblast. To je ten, který je na okraji triangulace a zároveň není hraničním tetrahedronem ze seznamu *H*. Dále je postup snadný. Tento tetrahedron označíme příznakem „vnější“. Nalezneme všechny trojúhelníky tohoto tetrahedronu, které jsou protnuté plochou tvořenou trojúhelníkem *abc*. Tetrahedrony sousedící přes tyto trojúhelníky zařadíme do zásobníku. Poté postupně vybíráme tetrahedrony ze zásobníku a provádíme stejné kroky. Pokud je tetrahedron již označen, nic neprovádíme, jen jej vyjmeme ze zásobníku. Pokud tetrahedron patří do hranice zasažené oblasti uložené v seznamu *H*, také nic dalšího neprovádíme. Takto pokračujeme až do vyprázdnění zásobníku.

Tímto postupem označíme vnější tetrahedrony. Tetrahedrony ze seznamu *S*, které nejsou označené příznakem „vnější“, tvoří zasaženou oblast. Ve skutečnosti nemusíme seznam *S* vytvářet. Ušetří se tím mnoho výpočetního času, protože nebudeme muset testovat všechny tetrahedrony na vzájemnou polohu vůči rovině. Seznam se používá jen pro nalezení vnitřku oblasti (na Obr. 6.5 je tímto vnitřkem prostor mezi hranami trojúhelníku *abc*). Vnitřek oblasti lze získat stejným způsobem, jakým označujeme vnější tetrahedrony. Začínáme v libovolném tetrahedronu ze seznamu *H* a používáme popsaný systém procházení tak, abychom stále zůstávali ve vnitřku oblasti. Tetrahedrony namísto označení příznakem můžeme přidávat do výsledného seznamu tetrahedronů zasažené oblasti.

Při implementaci metody však nastal problém. Při určité konfiguraci hranice algoritmus pronikl přes hranici do vnitřku oblasti, kterou označil jako vnější. Tato konfigurace je znázorněna právě na Obr. 6.5. Slabé místo, kde algoritmus může prolézt dovnitř, je v bodě *c*. Vnitřek a vnějšek oblasti zde není předělen uceleným pásem tetrahedronů. Proto algoritmus nenarazí na žádný tetrahedron a protnutým trojúhelníkem vejde do vnitřku oblasti, kterou poté označí jako vnější. Tento problém jsem se pokoušel odstranit pomocí různých podmínek testujících, zda bod je jeden ze tří vrcholů trojúhelníku. Toto sice vyřešilo problém, ale na druhou stranu se v některých případech neoznačily tetrahedrony, které se označit měly. Zasažená oblast obsahovala tetrahedrony, které obsahovat neměla. Tyto tetrahedrony se vyskytovaly v oblasti vrcholů povinného trojúhelníku *abc*.

Tato metoda se ukázala sice jako výpočetně celkem rychlá, na druhou stranu nedávala vždy korektní výsledky. Cílem je však vyzkoušet metody tvořící CDT v E^3. Hledání oblasti je úkol podružný, přesto může být závažným problémem, pokud není udělán přesně. Pokud by zasažená oblast obsahovala tetrahedrony navíc, je možné, že by to způsobilo špatné chování metody na CDT, která by jinak fungovala. Proto byla použita metoda brutální síly.

Metoda brutální síly

Metoda brutální síly je sice výpočetně náročná, ale dává zaručené výsledky, které jsou naší prioritou. Pracuje tak, že prochází všechny tetrahedrony protnuté rovinou vnuceného trojúhelníku *t*. Pro každý tento tetrahedron testuje každý jeho trojúhelník na průsečík s trojúhelníkem *t*. Pokud nějaký průsečík existuje, tetrahedron je zařazen do zasažené oblasti.

Metoda spolehlivě funguje pro veškeré možné případy. Příklad zasažené oblasti je uveden v příloze B.

Následující kapitoly se věnují algoritmům na konstrukci CDT v prostoru E^3. Nejprve je popsán algoritmus Maur rozšířený o jednu dimenzi.

6.3 Maur E3

Nyní bude popsána představa o tom, jak by metoda měla vypadat pro tři dimenze. Uvažme pouhé rozšíření o jednu dimenzi. Algoritmus rozvíjí postup popsaný v kapitole 3.3. Postupně budeme procházet jednotlivé kroky algoritmu a pro každý tento krok uvedeme rozšíření do prostoru E^3. Zde je potřeba říci, že žádný popis algoritmu Maur pro prostor E^3 neexistuje.

Nejprve se budeme věnovat nalezení zasažené oblasti. Pro další vysvětlování označme vnucený trojúhelník symbolem *t*.

Nalezení zasažené oblasti

Zasažená oblast je nalezena, jak je popsáno v kapitole 6.2. Pro algoritmus Maur E^3 předpokládáme, že bude postačovat přeměnit pouze zasaženou oblast.

Namapování bodů

Namapování bodů by vypadalo podobně jako v případě E^2. Body v prostoru E^3 se nebudou mapovat na paraboloid, ale paraboloid rozšířený o jednu dimenzi, čtyřrozměrný paraboloid. Jeho implicitní popis vypadá následovně:

$$0 = (x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 - q - w^2 - c - m\,x - n\,y - k\,z.$$

Rov. 14: Obecná rovnice čtyřrozměrného paraboloidu v E^3.

Význam značení je stejný jako v rovnici Rov. 2-4. Navíc je zde q, což je označení pro čtvrtou souřadnici dosazovaného bodu. Při hledání středů paraboloidu se hledal nejbližší bod ke vnucené hraně. Kritérium pro to, aby bod byl ze všech bodů nejbližší, bylo definováno tak, že kružnice opsaná hraničním bodům vnucené hrany a tomuto bodu nesmí obsahovat žádné další body z oblasti, kterou právě mapujeme. Střed této kružnice odpovídal prvním dvěma souřadnicím středu paraboloidu. Třetí souřadnice se dopočítala tak, aby se oba paraboloidy protínaly v místech vnucené hrany.

V prostoru E^3 se také hledá nejbližší bod. Ten je definován tak, že koule opsaná tomuto bodu a třem bodům vnuceného trojúhelníku nesmí obsahovat žádný další bod oblasti, kterou právě mapujeme. Princip zde je stejný. Při praktických testech by se používaly výpočty determinantů určující pozici bodu vzhledem ke kouli (vztah Rov. 10)). Posunutí paraboloidu ve 4. souřadné ose se vypočítá obdobně, jako je to popsáno v případě E^2. Problém je jen v tom, že situaci je skoro nemožné si představit. Principiální problém zde však není žádný.

Výsledkem tohoto kroku jsou váhy přiřazené bodům.

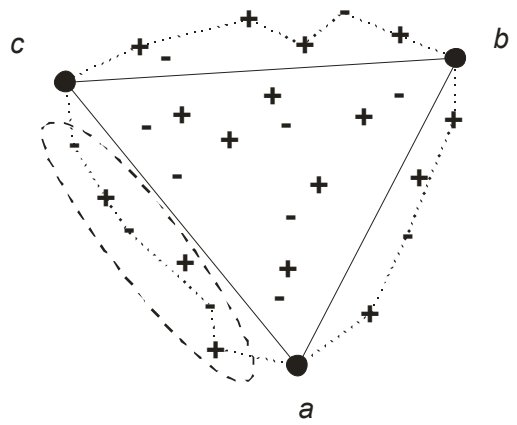
Konstrukce regulární triangulace

Existuje více způsobů, jak zkonstruovat regulární triangulaci v E^3. Regulární triangulace je často konstruována pomocí inkrementální metodou. Jedna taková metoda je popsána ve [fac95a]. Další metoda může být nalezena v [edel96a]. Regulární triangulace však přináší problémy, pokud se konstruuje neinkrementální metodou. V článku [joe89a] je dokázáno, že pro E^3 dokonce ani konstrukce Delaunayovy triangulace pomocí sekvence swapů vedoucích z libovolné triangulace nemusí vždy dojít k Delaunayově triangulaci. Při prohazování trojúhelníků je možné, že se algoritmus dostane do míst, kde se zasekne. Tím tedy ani konstrukce regulární triangulace pomocí lokálního prohazování nemusí vést k cíli. Toto můžeme tvrdit, protože Delaunayova triangulace je speciálním případem regulární triangulace. Regulární triangulace je přitom na problém zaseknutí náchylnější.

Hraniční tetrahedrony

Existuje jeden problém, pro který metoda Maur tak, jak je definována, nemusí fungovat. Metoda přiřazuje váhy tak, aby horní a dolní část zasažené oblasti nikdy nebyly propojeny hranou E^2 nebo trojúhelníkem v E^3. Při konstrukci E^2 CDT propojení oblastí opravdu nikdy nastat nesmí. Všechny body horní a dolní oblasti jsou propojeny přes hraniční body vnucené hrany. V E^3 situaci komplikuje, že toto propojení můžeme vyžadovat. Vysvětlení tohoto tvrzení poskytují následující odstavce.

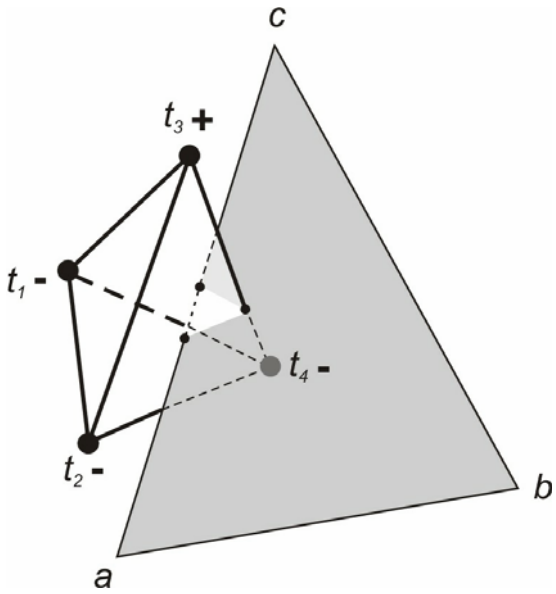
Mějme vnucený trojúhelník *t* tvořený body *a*, *b*, a *c*. Situaci znázorňuje Obr. 6.6. Zasažená oblast obsahuje body ležící nad trojúhelníkem (označené +) a body ležící pod trojúhelníkem (označené -).



Obr. 6.6: Oblast zasažená vnuceným trojúhelníkem *abc*. Znaménka + jsou body ležící nad trojúhelníkem, znaménka - znázorňují body ležící pod trojúhelníkem *abc*. Tečkovanou čárou je vyznačena obálka staré oblasti tetrahedronů promítnutá do směru pohledu. Čárkovaná elipsa zvýrazňuje jednu krajní skupinu bodů.

Obrázek znázorňuje zasaženou oblast při pohledu shora. Průmětem oblasti do roviny náčrtu je uzavřený polygon, který je na obrázku znázorněn tečkovanou čarou. Čárkovanou čarou je znázorněna jedna skupina krajních bodů ležících mimo trojúhelník (při pohledu shora). Před zrušením tetrahedronů zasažené oblasti byla situace taková, že horní a dolní krajní body spojovaly trojúhelníky. Tyto trojúhelníky sice protnuté zasaženým trojúhelníkem nebyly, ale patřily do tetrahedronu, který protnutý byl. Detailní pohled na jeden takový krajní tetrahedron schématicky znázorňuje Obr. 6.7.

Tetrahedron $t_1\ t_2\ t_3\ t_4$ je protnutý trojúhelníkem *abc*. Body t_1, t_2, t_4 leží pod trojúhelníkem (jsou označené znaménkem -), bod t_3 leží nad trojúhelníkem (označený znaménkem +). Trojúhelníky $t_1\ t_2\ t_3$ a $t_4\ t_2\ t_1$ protnuté nejsou, přesto tetrahedron spadá do zasažené oblasti, protože zbylé dva trojúhelníky protnuté jsou. Toto je slabé místo metody Maur. Ta totiž namapuje váhy tak, aby se mezi body horní a dolní oblasti nemohla vytvořit hrana trojúhelníku. Tak se však nevyplní prostor mezi body horní a dolní oblasti, protože je nemůže propojit žádný trojúhelník, tedy ani tetrahedron. Na okraji je třeba vytvořit tetrahedrony, které oblast vyplní. Vnější tvar oblasti musí zůstat po prohození tetrahedronů stejný jako před odstraněním původních tetrahedronů. Není tedy žádná jiná možnost, jak vyplnit oblast po tetrahedronu $t_1\ t_2\ t_3\ t_4$ jinak, než že budou existovat tetrahedrony, které budou mít trojúhelníky propojující obě oblasti. Ve výsledné triangulaci musí tedy být trojúhelník $t_1t_2t_3$. Ten ovšem podle metody Maur, jak je definována, nevznikne.



Obr. 6.7: Tetrahedron na hranici zasažené oblasti protnutý vnuceným trojúhelníkem abc. Obrázek je schématický, trojúhelník by byl ve skutečnosti pravděpodobně větší. Ostatní tetrahedrony zasažené oblasti znázorněny nejsou.

V prostoru E^2 takovýto problém nenastal. Tam totiž nebylo nutné oblasti navzájem propojit. V prostoru E^3 však tento problém nastává na okrajích zasažené oblasti. Ostatní tetrahedrony samozřejmě propojeny být nesmějí, protože by jejich hrany protínaly vnucený trojúhelník *t*.

Shrnutí metody MAUR E^3

K tomu, aby algoritmus fungoval, je zapotřebí, abychom dokázali nalézt zasaženou oblast, přiřadit bodům váhy a provést regulární triangulaci. Jak již bylo napsáno, první dva kroky lze provést, není žádný principiální problém, který by tomu bránil. U provedení regulární triangulace je však situace o dost horší. Při použití metody lokálního prohazování trojúhelníků riskujeme, že se proces zasekne. Řešením by mohlo být použití některého z principů podobných prioritní frontě popsané v algoritmu inkrementálního vkládání hran, viz kapitola 6.1.1. Problém je, že pro metodu Maur neznáme přesný postup v určování priority swapů. Z dosavadních zkušeností nejsme ani schopni takový postup vynalézt.

Další otázkou je, zda bude stačit pouze přeměnit konfiguraci tetrahedronů v zasažené oblasti. Ve variantě E^2 to bylo dostačující. Na to bohužel nedokážeme s určitostí odpovědět. Algoritmus inkrementálního vkládání hran používá pouze tuto oblast. Otázkou je však, zda bude stačit pouze zasažená oblast také algoritmu Maur. Tato otázka se dá vyřešit jen návrhem metody a jejím praktickým otestováním.

Poslední problém, který byl popsán, je problém s tetrahedrony na krajích zasažené oblasti. Metoda tak, jak je popsána, by nevyplnila krajní oblast. Řešením by mohlo být zavést do algoritmu testy, které vyloučí prohození trojúhelníků na okrajích zasažené oblasti přesto, že váhy bodů si vynucují jejich prohození. Otázkou je, zda by to nějak neovlivnilo chod algoritmu, zda by byla zaručena jeho konvergence, a zda by algoritmus dával korektní výsledky.

Při zvážení všech vlastností, které jsme uvedli, jsme se rozhodli metodu v E^3 neimplementovat. Je sice pravda, že veškeré problémy by se daly nějakým způsobem řešit, ale pak by se jednalo o příliš komplikované řešení. Zároveň bychom stále neměli zaručeno, že metoda bude fungovat. Implementace algoritmu Maur by byla časově velmi náročná a

pravděpodobně by nemohla být včas dokončena. Proto se ukázalo vhodnější zvolit nějakou jinou metodu.

Přesevše je metoda Maur zajímavým řešením. Používá totiž stejné principy, jako existující algoritmus inkrementálního vkládání hran (viz kapitola 6.1.1), který funguje bez problémů. Bezproblémová funkčnost je však zajištěna i dalšími vlastnostmi algoritmu.

Následující podkapitola se věnuje algoritmu založenému na lokálním prohazování trojúhelníků.

6.4 Lokální prohazování trojúhelníků

Pro implementaci byl vybrán algoritmus lokálního prohazování trojúhelníků. Důvody pro zvolení tohoto algoritmu jsou následující. Algoritmus je jednodušší na implementaci než ostatní algoritmy. Zde je potřeba říci, že jakýkoliv algoritmus pracující s tetrahedronovou sítí je složitý a časově náročný na implementaci. Dalším důvodem je to, že algoritmus je jednoduše upravitelný a rozšiřitelný o další vylepšení.

Metoda prohazování trojúhelníků nemusí však fungovat vždy, jak již bylo řečeno dříve. Můžeme narazit na situaci, kdy se algoritmus zasekne. Naše představa je však taková, že tato situace by měla vznikat ojediněle. Cílem je toto prakticky ověřit. Pokud nastane případ, kdy se metoda zasekne, pokusíme se výskyt tohoto případu minimalizovat, aby byla metoda použitelná. Při výkladu metody předpokládáme, že CDT k danému vstupu existuje.

Algoritmus je založený na principu vnucení trojúhelníku pomocí lokálního prohazování trojúhelníků. Algoritmus je rozšířením E^2 varianty algoritmu S. W. Sloana popsaného v kapitole 3.2. Přesný popis algoritmu může být nalezen v [sloan93a]. Algoritmus přeměňuje zasaženou oblast pomocí swapů tak dlouho, dokud se v zasažené oblasti neobjeví tetrahedron obsahující trojúhelník *abc*. Poté následuje druhá část algoritmu a obnova Delaunayových vlastností zasažené oblasti. Nyní popíšeme první část algoritmu a to vnucení povinného trojúhelníku *abc*.

Nejprve jsou nalezeny všechny trojúhelníky protnuté vnuceným trojúhelníkem *abc*. Povinný trojúhelník označme *t*. Tyto trojúhelníky uloží do seznamu označeného `TrojList`. Algoritmus prochází trojúhelníky z `TrojList` a hledá pro ně příslušné swapy. Pokud swap není možný, zařadí trojúhelník na konec seznamu `TrojList`. Pokud swap možný je, provede prohození. Při prohození zanikají staré tetrahedrony a nové tetrahedrony vznikají. Je proto potřeba aktualizovat zasaženou oblast. Zároveň je potřeba aktualizovat seznam Trojúhelníků `TrojList`.

Postup je popsán následujícím algoritmem. Význam proměnných je popsán v tabulce Tab. 1.

```
AffArea = Najdi zasaženou oblast trojúhelníkem t;

TrojList = Vytvoř seznam trojúhelníků, které jsou protnuty trojúhelníkem t
            a patří tetrahedronům z AffArea;

While TrojList není prázdný
Begin
      tri = TrojList.VyjmiTrojúhelník();
      swap = najít vhodný swap pro tri;

      If (swap není možný)
      Begin
            TrojList.VložNaKonec(tri);
            continue;   /* Pokračuj v další iteraci while */
      End
```

```
        Proveďswap(swap);

        /* Aktualizace seznamů trojúhelníků */
        List zasaženéTroj = Najdi trojúhelníky tetrahedronů zasažené swapem;
        TrojList.Odstran(zasaženéTroj);
        List novéTroj = Najdi všechny trojúhelníky nových tetrahedronů
                        protnuté trojúhelníkem t;
        TrojList.VložNaKonec(novéTroj);
        Aktualizuj zasaženou oblast AffArea: vyjmi staré TH a vlož nové;
End
```

Alg. 2: Prohazování trojúhelníků

Tab. 1: Proměnné algoritmu a jejich význam

Typ	Název	Význam
List	`TrojList`	Seznam všech trojúhelníků protnutých zasaženým trojúhelníkem *t*.
Množina	`AffArea`	Zasažená oblast. Množina všech tetrahedronů zasažené oblasti.
Trojúhelník	`tri`	Aktuálně zpracovávaný trojúhelník.
Swap	`swap`	Typ swapu, který má být proveden. Nese informace i o tetrahedronech, které swap používá.
Množina	`zasaženéTroj`	Množina všech trojúhelníků patřících tetrahedronům, na kterých byl proveden swap.
Množina	`novéTroj`	Množina všech trojúhelníků patřící novým tetrahedronům. Do této množiny jsou uloženy jen trojúhelníky, které jsou protnuté trojúhelníkem *t*.

Nalezení vhodného swapu k trojúhelníku je provedeno následovně. Naleznou se dva tetrahedrony, které trojúhelník sdílí. Dále se zjistí, zda má tato konfigurace konvexní tvar. Pokud ano, je proveden swap 23, pokud ne, swap 32[18]. To, že swap není možný, znamená, že nastal případ, kdy swap 32 potřebuje třetí tetrahedron, který však neexistuje. Pokud se tak stane, algoritmus zařadí trojúhelník na konec seznamu a pokračuje v další iteraci.

Po každém swapu je zapotřebí aktualizovat používané seznamy a množiny. Je to tím, že při swapu tetrahedrony zanikají a vznikají nové. Takto je potřeba udržovat aktuální seznam všech protnutých trojúhelníků a množinu tetrahedronů tvořící zasaženou oblast.

Po vyprázdnění seznamu `TrojList` je do triangulace vnucen trojúhelník *t*. Oblast je trojúhelníkem rozdělena na dvě. Přesto stále nesplňuje podmínky CDT, protože tetrahedrony oblasti nejsou Delaunayovské. Obnovení Delaunayova kritéria pro tetrahedrony v oblasti probíhá podobně jako její rozdělení na dvě části pomocí vnucení trojúhelníku. Algoritmus zde uveden nebude, protože je velmi podobný algoritmu Alg. 2.. Jediný rozdíl je v kritériu, podle kterého se hledají trojúhelníky. Na místo seznamu protnutých trojúhelníků se vkládají trojúhelníky, které nesplňují Delaunayovo kritérium lokálně optimálního trojúhelníku (viz. definice 2-5). Nikdy se nesmí z triangulace odstranit trojúhelník *t*. Tento trojúhelník se při testech přeskakuje a nikdy není vkládán do seznamu trojúhelníků. Další rozdíl je v testování,

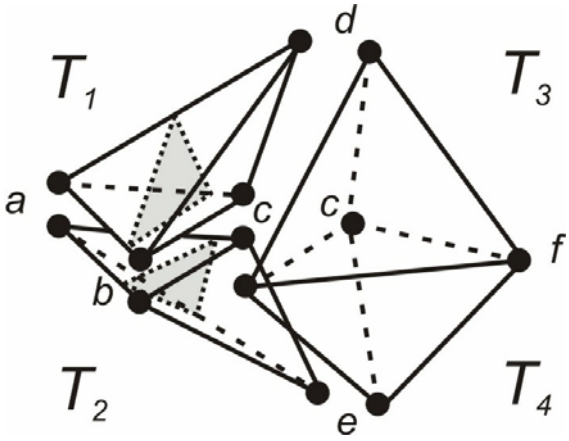
[18] Detailně jsou swapy (prohození konfigurace tetrahedronů) rozebrány v kapitole 5.3. Zároveň jsou tam popsány podmínky, za kterých lze určitý swap provést.

zda je swap možný. Pravidla platí stejná, jen navíc je přidána podmínka, že swap nesmí porušit trojúhelník *t,* jenž byl vnucen v první části algoritmu.

Chování metody

Algoritmus byl naimplementován podle návrhu. Metoda prohazování trojúhelníků se však ukázala být nepoužitelná. Metoda byla schopná zkonstruovat CDT pouze pro malé zasažené oblasti. Při větších oblastech byla úspěšnost menší.

Důvod, proč metoda nevytvoří CDT, je následující. Jak metoda provádí swapy, stále více tetrahedronů se přeměňuje tak, že leží mimo protnutý trojúhelník. V některých případech je metoda úspěšná a všechny tetrahedrony se uskupí tak, že v zasažené oblasti vznikne vnucený trojúhelník *t*. V opačném případě v oblasti zbývá několik tetrahedronů, které dál již nemohou být prohozeny tak, aby trojúhelník mohl vzniknout. Swapy, které by byly potřeba, nelze provést. Jedná se totiž o swapy 32, kterým chybí třetí tetrahedron, jak je to popsáno v kapitole 5.3. Situace je znázorněna na Obr. 6.8. Místo třetího tetrahedronu jsou v triangulaci dva tetrahedrony (na obrázku jsou to T_3 a T_4). Tyto tetrahedrony však již nejsou protnuté rovinou trojúhelníku *t* a proto se jejich konfigurace nikdy nezmění (algoritmus již s nimi nepracuje). Swap tedy nelze provést a zařazení trojúhelníku na konec seznamu není řešením, protože konfigurace tetrahedronů v místě chybějícího tetrahedronu se již nezmění.



Obr. 6.8: Swap 32. Tečkovanou čarou je znázorněn průsečík s vnuceným trojúhelníkem *t*. Tetrahedrony T_1 a T_2 jsou trojúhelníkem protnuty.

V experimentech nastávají dva typy případů. V prvním se algoritmus zacyklí, protože nemá, co provést. Stále cyklicky hledá swapy, ale žádný swap není již možný. Druhý případ je, že nějaké swapy stále možné jsou, ale nevedou k cíli. Algoritmus provádí tyto swapy tam a zpět. Oba dva případy jsou v podstatě stejné. Nelze provést swap, který by mohl konfiguraci nějak změnit. Algoritmus tedy došel do místa, které je pro něj slepou uličkou.

Přitom není řešením povolit přibrat do seznamu tetrahedrony, které nám chybí (na Obr. 6.8. jsou to T_3 a T_4). Je to tím, že případy, kdy potřebujeme třetí chybějící tetrahedron, jsou velmi časté. Metoda by nekonvergovala, protože by tak často vracela zpět opravenou oblast, takže ve výsledku by nikam nepostupovala. I tento případ jsme experimentálně vyzkoušeli.

Pro odstranění tohoto problému jsme zkusili metodu upravit tak, že někdy zařadí do zásobníku tetrahedrony, které již nejsou protnuté. Tyto tetrahedrony potom prohodí. Tetrahedrony do zásobníku zařadí pouze jednou za stanovený počet iterací (například jednou za 50 iterací). Tím odstraníme výše uvedený problém nekonvergence metody, kdy metoda nikam nepostupuje, protože stále rozrušuje již hotovou oblast. Ani toto nebylo řešení, které by vedlo k cíli. Metoda se po chvíli znovu zasekne.

Doplnění o Delaunayovy testy

Kvůli předchozím neúspěchům jsme se rozhodli metodu rozšířit o Delaunayovy testy tak, aby trojúhelníky, které vznikají, splňovaly Delaunayovo kritérium. Znamená to, že oblast, která vzniká, bude již splňovat Delaunayovo kritérium bez použití druhé části metody. Přesný postup je následující. Mějme seznam S všech tetrahedronů, které nejsou protnuté vnuceným trojúhelníkem t. Postupně během iterací prohazování se do tohoto seznamu zařazují tetrahedrony, které jsou již v pořádku. Iterace poté vypadá následovně. Provedeme swap, jak bylo již bylo popsáno v algoritmu Alg. 2.

Po prohození najdeme nově vzniklé tetrahedrony, které již nejsou protnuté vnuceným trojúhelníkem t. Pokud koule opsaná tomuto tetrahedronu obsahuje nějaký bod sousedících tetrahedronů, které jsou již v seznamu S, swap je neplatný a provedeme zpětné prohození. Po zpětném prohození je situace stejná jako před swapem. Aktuálně testovaný trojúhelník vložíme na konec seznamu pro pozdější zpracování. Druhou variantou je předem otestovat, zda konfigurace po swapu bude Delaunayovu podmínku splňovat. Pokud ne, swap se neprovede. Pro implementaci je však jednodušší swap provést a poté zase vrátit, než testovat podmínky na konfiguraci tetrahedronů, kterou nemáme a musíme ji nejprve vypočíst (což je ve své faktické podstatě provedení swapu).

Hlavní myšlenkou této modifikace metody je zamezení vzniku tenkých tetrahedronů, které by mohly vést k zacyklení algoritmu. Vše, co bude vznikat, budou pouze Delaunayovy tetrahedrony. Domníváme se, že tyto tetrahedrony budou mít příznivé vlastnosti pro konvergenci metody.

Ani toto rozšíření však nepřineslo žádné zlepšení. Základní problém metody bez testů je ten, že pokud metoda zvolí špatnou cestu swapů, nedojde k cíli. Řešením by mohlo být pořadí swapu nějakým způsobem určovat, aby vedly k cíli. Delaunayovy testy by mohly být oním omezením, které by dovedlo algoritmus k cíli. Experimenty ukázaly, že to tak není. Jen velice málo případů je těch, kdy rovnou vzniká Delaunayova triangulace. Je to velké omezení, při kterém metoda každý swap, který provede, hned vrátí zpět. Takto omezená metoda se zasekne daleko dříve než metoda bez testů, protože nemůže provést žádný swap, který produkuje Delaunayovy tetrahedrony.

Zhodnocení metody

Metoda se ukázala jako nepoužitelná. Před experimenty jsme předpokládali podstatně vyšší úspěšnost metody. V pokusech jsme zašli dále než jsme původně zamýšleli, ale i po všech experimentech se nepodařilo najít vhodnou modifikaci pro zvýšení úspěšnosti metody na přijatelnou mez.

Možné další řešení by mohlo být následující. Ukázalo se, že pokud se ze seznamu vybírají trojúhelníky náhodně, metoda má šanci k cíli dospět. To znamená, že existují triangulace, kde lze CDT pomocí lokálního prohazování trojúhelníků vykonstruovat, jen je zapotřebí zvolit správné pořadí swapů. Problém je najít správné pořadí swapů, aby metoda vedla k cíli. Jedno řešení by mohlo být pomocí vytvoření grafu swapů. Každý uzel grafu by představoval určité uspořádání tetrahedronů v zasažené oblasti. Hrany mezi nimi by představovaly swapy, kterými lze přeměnit oblast z jednoho uspořádání na druhé. Procházením tohoto grafu by bylo možné dojít k cílenému uspořádání.

Otázkou však je, zda pro všechna uspořádání existuje nějaká cesta swapů, která je oblast přemění do oblasti cílové, v které existuje trojúhelník t. Jak bylo již napsáno výše, v prostoru

E^3 není zaručeno, že existuje posloupnost swapů, která převede libovolnou triangulaci na Delaunayovu triangulaci[19]. Podobný problém může nastat i v tomto případě, a to ten, že neexistuje vždy možnost pomocí swapů vnutit trojúhelník *t*. Je tedy důležité určit přesněji důvod nekonvergence algoritmu. Důvod může být ten, že sice existuje posloupnost swapů, která oblast úspěšně přemění na cílovou oblast, ale algoritmus takovou cestu nenašel. Druhá možnost je, že taková posloupnost swapů vůbec neexistuje. Z experimentů je však více než pravděpodobné, že se při vykonávání vyskytují oba problémy najednou v závislosti na případech. Více bude popsáno v kapitole 8.

Volení správného pořadí swapů řeší inkrementální algoritmus z kapitoly 6.1.1 pomocí prioritní fronty. Takto navrženou frontu však zde nelze použít, protože inkrementální algoritmus je založen na jiném principu, s nímž je fronta úzce spojena. Bylo by nutné přijít na způsob jak frontu navrhnout pro tuto metodu.

Další možné řešení tohoto problému by mohlo být dělení vnuceného trojúhelníku. Tato úvaha vychází z faktu, že metoda je úspěšnější pro malé oblasti. Trojúhelník je možné rozdělit vložením bodu do triangulace. V triangulaci tak vzniknou další tetrahedrony. Vložený bod označme *p*. Namísto vnucování trojúhelníku *abc* se budou postupně do triangulace vnucovat trojúhelníky *abp*, *bcp* a *cap*. Každý z těchto dílčích trojúhelníku bude vytvářet menší zasaženou oblast. Tím bude pravděpodobnost úspěchu metody vyšší.

Metoda se neukázala být stoprocentně úspěšnou. V kapitole 8 jsou výsledky detailně popsány a možná rozšíření jsou podložena provedenými experimenty. Pro možnost porovnání metody s jiným přístupem vnucení trojúhelníku byla navržena další metoda, metoda rekonstrukce zasažené oblasti.

6.5 Metoda rekonstrukce zasažené oblasti

Tato metoda je založena na znovuvytvoření zasažené oblasti. Algoritmus z triangulace vyjme všechny tetrahedrony zasažené oblasti, vloží vnucený trojúhelník a poté vytvoří celou oblast znovu. Postup je podobný algoritmu Anglada z kapitoly 3.1. Tato metoda je opět jistým rozšířením metody z prostoru E^2 do prostoru E^3. Metoda, jak je popsána, je navržena námi, nečerpá tedy z žádného zdroje. Cílem je zjistit, zda takový přístup může v prostoru E^3 fungovat.

Pro vysvětlování algoritmu používejme označení *t* pro vnucený trojúhelník *abc*.

Naše úvaha je ta, že tomuto postupu postačí pouze zasažená oblast. Vyplývá to z toho, že zasažená oblast je postačující i pro CDT v E^2. Algoritmus začíná tím, že z triangulace vyjme všechny tetrahedrony zasažené oblasti. Tím vytvoří v triangulaci díru. Poté vytvoří tetrahedron, který se skládá z bodů *a*, *b*, *c* a bodu, který leží nejblíže trojúhelníku *abc*. Nejbližší bod se hledá stejným způsobem, jako již bylo popsáno u algoritmu Maur v kapitole 3.3, kde je rozebírán rovinný případ, nebo v kapitole 6.3 pro prostor E^3. V kapitole 3.3 je navíc popsán přesný popis postupu nalezení tohoto bodu a je možné jej při implementaci použít. Hledáme tedy bod *x*, kde koule opsaná tomuto bodu *x* a trojúhelníku *abc* nebude obsahovat žádné další body ležící stejným směrem od trojúhelníku jako bod *x*. Po nalezení tohoto bodu vytvoříme tetrahedron *abcx*. Tím v triangulaci vznikl trojúhelník $t = abc$. Zároveň vznikly tři nové trojúhelníky u vloženého tetrahedronu. Stejný postup opakujeme rekurzivně i pro nové tři trojúhelníky. Tak vznikají nové trojúhelníky, které rekurzivně dále

[19] Problém je rozebírán v [joe89a].

zpracováváme. Tímto postupem algoritmus postupně zaplňuje oblast. Rekurzivní postup se zastaví ve chvíli, kdy narazíme na hranici zasažené oblasti nebo na již zpracovanou oblast.

Toto je jen stručný popis pro vytvoření hrubé představy, problém je však o něco složitější. Následující text popisuje přesněji algoritmus na vytvoření oblasti. Algoritmus je popsán v bodech namísto pseudokódu. Je zde popsán postup vnucení jedné hrany. Algoritmus používá frontu *F*, do níž vkládá postupně trojúhelníky, z kterých budou vytvořeny nové tetrahedrony. Nejprve popíšeme postup bez detailu, jak je tetrahedron vytvořen. Tento první popis označme **A**.

A:

1. Nalezení zasažené oblasti. Postup nalezení oblasti je popsán v kapitole 6.2.
2. Každý bod zasažené oblasti se vloží do seznamu *P*. Trojúhelníky tvořící obálku zasažené oblasti vložíme do seznamu *O*.
3. Vyjmutí tetrahedronů zasažené oblasti z triangulace.
4. Vnucený trojúhelník *t* se vloží do prázdné fronty *F*.
5. Dokud není fronta *F* prázdná, opakujeme následující vnořené body:
    a. Vyjmeme trojúhelník z fronty. Označme jej *troj*.
    b. Vytvoříme tetrahedron, označme jej *th,* z trojúhelníku tak, aby čtvrtý bod ležel ve směru normály. Tento krok bude detailně popsán později v části **B**. Pokud tetrahedron nemohl být vytvořen, přejdeme do bodu 3a. Pokud byl vytvořen, přidáme jej do triangulace.
    c. Každý trojúhelník z tetrahedronu *th* kromě trojúhelníku *troj* vložíme do fronty *F*. Pokud se jedná o vnucený trojúhelník *t,* nevkládáme ho. Zároveň nevkládáme trojúhelník, který odpovídá trojúhelníku obálky zasažené oblasti, které jsme v bodu 2 uložili do seznamu *O*. Před vložením trojúhelníku je třeba se ujistit, zda trojúhelník je správně orientovaný, aby při pozdějším zpracování byl vytvářen tetrahedron *th* správným směrem (a ne dovnitř už vytvořeného trojúhelníku). Trojúhelník tedy orientujeme tak, aby normála směřovala ven z tetrahedronu[20].
6. Nyní je fronta prázdná. Je potřeba ještě rekonstruovat oblast na druhé straně vnuceného trojúhelníku. To provedeme tak, že do fronty zařadíme vnucený trojúhelník *t* orientovaný opačně a opakujeme iterační proces skokem do bodu 5. Někdy tento postup není nutné provádět. Je to proto, že obě oblasti se vyplní najednou. To proto, že algoritmus proleze do druhé oblasti skrz prostor mezi krajem vnuceného trojúhelníku a hranicí zasažené oblasti. Tento prostor je popisován v algoritmu Maur v kapitole 6.3 a je znázorněn na Obr. 6.6. Přesto, že je možné, že tetrahedrony tímto směrem již existují, postup vykonáme. Algoritmus se zastaví hned na počátku, když nebude moci vytvořit žádný tetrahedron. V opačném případě vyplní druhou část oblasti.

Tímto je oblast obnovena. Je však třeba popsat ještě bod 5b, vytváření tetrahedronu. To probíhá následovně.

[20] Může směřovat i dovnitř. Je však pak zapotřebí změnit i vytváření tetrahedronu. Jediné, co je důležité, aby algoritmus s orientacemi bodů vhledem k rovinám zacházel stejně.

B: (detailní popis bodu 5b)

Vstupem je orientovaný trojúhelník *troj*.

1. Ověříme, zda tetrahedron ještě neexistuje. To provedeme tak, že prohledáme novou oblast a pokud najdeme dva tetrahedrony, které obsahují trojúhelník troj, tetrahedron je již vytvořen. V tom případě vrátíme příznak, že tetrahedron nebyl vytvořen.
2. Vytvoříme seznam *P'*, který bude obsahovat body ze seznamu *P* (vytvořeného v bodě **A**:2). Do seznamu *P'* vložíme pouze body, které jsou od trojúhelníku orientovány stejným směrem, jako jeho normála. Zároveň tyto body seřadíme podle vzdálenosti od trojúhelníku *troj*. Testování vzdálenosti je popsáno na začátku této podkapitoly. Znamená to, že koule opsaná prvnímu bodu ze seznamu *P'* a trojúhelníku *troj* nebude obsahovat žádný jiný bod ze seznamu *P'*[21]. Koule opsaná druhému bodu ze seznamu *P'* a trojúhelníku *troj* bude obsahovat pouze první bod a tak dále.
3. Pro všechny body ze seznamu *P'* provádíme následující akce. Aktuálně vybraný bod označme *p*.
    a. Vytvoříme dočasný tetrahedron, označme jej *dth* tvořený trojúhelníkem *troj* a bodem *p*.
    b. Tetrahedron *dth* je třeba otestovat, zda neprotíná jiný tetrahedron. Varianta s užitím postupu brutální síly je testovat *dth* na průsečík s každým tetrahedronem dosud vzniklým v zasažené oblasti. Pokud existuje nějaký průsečík, pokračujeme s dalším bodem vybraným ze seznamu *P* krokem 3a. Toto je velice důležitý krok, který nemůže být vynechán. Kdyby nebyl proveden, vzniklé tetrahedrony by se mohly navzájem křížit. Je to tím, že i nejbližší bod k trojúhelníku *troj* může s trojúhelníkem tvořit tetrahedron, který bude křížen jiným již existujícím tetrahedronem. Tento případ může nastat, když algoritmus narazí na oblast již vytvořených tetrahedronů.
    c. Tetrahedron je v pořádku. Vrátíme jej do nadřazeného algoritmu **A**.
4. Žádný vhodný tetrahedron nebyl vytvořen (kdyby byl, algoritmus by skončil v bodě 3a). Vrátíme vhodný příznak.

Tímto postupem vytvoříme tetrahedron k danému trojúhelníku. Pokud to není možné, vrátíme příznak, že tetrahedron nebyl vytvořen.

Takto navržená metoda je celkem výpočetně náročná kvůli hledání průsečíků tetrahedronu s jinými tetrahedrony. Hledání průsečíků je možné určitým způsobem redukovat jen na tetrahedrony, které jsou v blízkosti nově tvořeného tetrahedronu.

Metoda na rozdíl od předchozí metody má zaručeno, že vždy vnutí do triangulace trojúhelník *t*. Problém však může být ten, že oblast bude vytvořena špatně a v triangulaci budou vznikat díry a nebo tetrahedrony, které se navzájem kříží. Jakákoliv numerická chyba například při testování průsečíků způsobí, že vznikne jeden chybný tetrahedron, který se bude křížit s jiným. Druhá možnost je, že tetrahedron, který vzniknout měl, nevznikne a v triangulaci bude díra.

Metoda rekonstrukce zasažené oblasti nepřinesla příznivé výsledky. I zde se v některých případech podařilo rekonstruovat oblast a v některých ne. Jak se ukázalo během experimentů,

[21] Koule bude obsahovat ve svém vnitřku body ze zasažené oblasti. Ty jsou ale na opačné straně od trojúhelníku *troj*, proto nejsou v seznamu *P'*.

metoda není tolik náchylná na velikost zasažené oblasti, jako metoda předchozí. Na druhou stranu je v průměru méně úspěšná než metoda lokálního prohazování trojúhelníků.

Metoda vždy v triangulaci vytvoří vnucený trojúhelník *t*. Na druhou stranu v mnoha případech vznikne v triangulaci díra. To znamená, že se zasažená oblast nevyplní celá novými tetrahedrony, ale zůstane v ní nějaký volný prostor. Nedokážeme s jistotou říci, co zapříčiňuje problém. Jedna možnost je ta, že problém zaviňují numerické nepřesnosti. U metod tohoto typu je to častý problém. Druhý problém může být principiálního charakteru. To znamená, že takto navržený algoritmus nemusí vést k cíli. Otázkou je, zda může nastat situace, kdy není možné do triangulace vytvořit další tetrahedron, přestože je třeba zaplnit vnitřní prostor. Na tuto otázku bohužel nedokážeme z dosažených znalostí odpovědět.

Metoda byla nejprve navržena a vyzkoušena bez testů na průsečíky, což vedlo k tomu, že se tetrahedrony křížily a tvořily se tetrahedrony tam, kde být neměly. Ukázalo se, že je třeba testovat průsečíky tetrahedronů. Situace je již jednou vysvětlena výše v algoritmu. Další problém je ten, že tetrahedrony vznikaly tak, že obsahovaly tetrahedrony ve svém vnitřku. Algoritmus konstrukce se většinou ani nezastavil a stále tvořil tetrahedrony obsahující tetrahedrony vytvořené v předchozích krocích. Testování průsečíků pozměnilo metodu tak, že začala dávat smysluplné výsledky. Přesto se nepodařilo metodu vylepšit tak, aby konstruovala spolehlivou síť bez vnitřních děr.

Metoda se ukázala být nestabilní a časově velmi náročná na provedení. Drobná numerická chyba nebo chyba při implementaci vede k závažné chybě v triangulaci. Toto se u jiných metod neobjevuje. U většiny metod chyba způsobí například, že vznikne oblast, kde není přesně dodrženo Delaunayovo kritérium. Nebo se dá chyba identifikovat a provést vhodné opravné opatření. Tím může být například rozdělení vnucovaného trojúhelníku na více částí. Rozdělením se mohou odstranit numerické problémy. Pokud ovšem vznikají v triangulaci díry nebo tetrahedrony, které se navzájem kříží, je to velmi nepříjemný a zásadní problém, který se velmi obtížně opravuje.

Výsledkem testování této metody je zjištění, že metoda není vhodná pro konstrukci CDT.

7 Implementace

V této kapitole bude stručně popsána provedená implementace řešení Maur v prostoru E^2 a metod v prostoru E^3. Při popisování implementace budou vynechány detaily (např. názvy metod). Budou popsány jen základní principy. Pro detailnější popisy doporučuji prohlédnout zdrojový kód, který je přehledně komentován.

Veškerá implementovaná řešení byla provedena v Microsoft Visual Studiu 2005, v jazyce C#.

7.1 CDT E^2

Nejpodstatnější z implementace jsou datové struktury, které byly užity. Triangulace je uložena pomocí dvou seznamů: seznamu bodů a seznamu trojúhelníků.

Point (bod)

Bod je implementován jako objekt. Jeho jediné tři položky jsou souřadnice *x*, *y* a váha bodu *w*. Body jsou uloženy v poli bodů, každý bod má svůj index, který je daný jeho pořadím v poli.

Triangle (trojúhelník)

Trojúhelník je implementován jako objekt. Všechny trojúhelníky jsou uloženy v poli trojúhelníků. Každý trojúhelník má svůj index, který je dán jeho pořadím v tomto poli. Trojúhelník obsahuje indexy tří bodů, z kterých se skládá. Indexy odkazují do pole bodů. Zároveň uchovává své tři sousedy jako tři indexy trojúhelníků. Struktura již dále není dělena na hrany, protože to není potřeba. Přesto je potřeba u každé hrany uchovat, zda je povinná. Trojúhelník obsahuje tři hodnoty `true` nebo `false`, které určují, zda je hrana povinná. Jedná se o pole tří `boolean` hodnot, pořadí v poli určuje o kterou hranu se jedná. Pokud jsou v poli tři body *a*, *b*, *c*, potom hrany odpovídající tomuto pořadí bodů jsou *bc*, *ca*, *ab*. Na stejnou pozici jakou má bod se tedy do jiného pole uloží jeho protější hrana. Toto je standardní postup ukládání podobných dat. Stejný princip platí i pro pořadí sousedů, kde se k bodu se váže jeho protější trojúhelník.

Základní princip ukládání struktur je tedy přes indexy objektů v poli. Uložení odpovídá formátu vstupních souborů, které jsou organizovány stejným způsobem. Program používá další objekty, které budou stručně vysvětleny.

Circle (kružnice)

Jedná se o strukturu, která uchovává data o nalezené kružnici. Jejími prvky jsou souřadnice *x*, *y* a poloměr *r*.

Edge (hrana)

Tato struktura se používá pro reprezentaci povinné hrany v průběhu vnucování hrany. Obsahuje pouze index bodu *a* a *b*.

Vector2

Reprezentuje dvojrozměrný vektor. Vektory jsou velmi často používány. Kromě základních směrových složek *x* a *y* vektor obsahuje metody na normalizaci vektoru a přetížený operátor na výpočet skalárního součinu.

Program vytváří Delaunayovu triangulaci pomocí knihovny `DTlib.dll`[22]. Knihovna je vytvořena v Borland Delphi 7.0. Knihovna je tedy psána v neřízeném kódu. Bylo zapotřebí vytvořit rozhraní jak uvnitř knihovny tak i v programu pro vzájemnou komunikaci mezi řízeným a neřízeným kódem. Rozhraní je naprogramováno uvnitř metody `Delaunay()`.

V průběhu implementace byly vytvářeny testy, které kontrolovaly konzistenci dat. To bylo provedeno pomocí `assert` funkce, která zahlásí chybu, pokud její vstupní podmínka není splněna. V klíčových částech kódu byly tyto testy používány. Ukázaly se být nezbytnou pomůckou. Pomáhají odhalit většinu chyb. Čím více je v programu testů, tím je jednodušší hledání chyb. Problém je, že každá chyba se nemusí projevit hned. Pokud se například někde prohodí indexy bodů a stane se to pouze u jednoho trojúhelníku ze sta, chyba se může projevit až při dalším zpracování sítě. Pak je problém zjistit, kde přesně chyba vznikla. Testy pomáhají ihned chybu identifikovat. Bez testů by bylo psaní takového kódu velmi obtížné. Jen pro představu, zdrojový kód má asi 2500 řádek (E^3 varianta dokonce 5000). Chyba každého řádku může způsobit vážné chyby v chodu programu. Testy tyto chyby efektivně odhalují.

Implementace je místy provedena metodami hrubé síly. Cílem diplomové práce nebylo psát optimalizovaný kód. Optimalizace kódu bývá většinou provedena až ve chvíli, kdy víme, že algoritmus funguje. U algoritmu Maur jsme toto dopředu nevěděli.

Pro zobrazení trojúhelníkových sítí jsem používal vizualizér CDT[23].

7.2 CDT E^3

Varianta E^3 je velmi podobná E^2 variantě. Používá stejné principy programování. Proto zde nebude detailně rozebírána. Jediný podstatný rozdíl je v datový strukturách.

Přístup ukládání dat přes indexy do polí má výhodu kompatibility se standardními přístupy ukládání trojúhelníkových sítí. Na druhou stranu má i své nevýhody. Jedna z nich je, že je potřeba vytvořit hodně zdrojového kódu. Zdrojový kód se zároveň stává nepřehledným, jelikož vše je komplikovaně odkazováno přes indexy do polí. To vede k nárůstu chyb ve zdrojovém kódu. Další nevýhodou je debugování takového kódu. Pokud například víme, že trojúhelník má tři sousedy s indexy 2, 87 a 2036, a chceme se na tyto trojúhelníky podívat, musíme je nejprve najít v seznamu trojúhelníků.

Datová struktura navržená pro E^3 variantu tento problém řeší. Vše je řešeno přes reference na objekty. Tetrahedron odkazuje pomocí referencí na čtyři body. Sousedi jsou také uloženi jako reference. Jediné, co je tedy jiné oproti předchozímu řešení je to, že se vyřadilo pole, přes které vše bylo indexováno. Práce s novou strukturou byla o dost jednodušší. Aby se zachovala kompatibilita se systémem indexování, každý tetrahedron a bod nese své identifikační číslo, které odpovídá jeho v pořadí v poli, z kterého byla data načtena. Tím je možné strukturu

[22] Knihovna byla napsána Doc. Dr. Ing. Ivanou Kolingerovou.

[23] Vizualizír byl vytvořen Doc. Dr. Ing. Ivanou Kolingerovou.

uložit zpět do pole nebo do souboru. Toto identifikační číslo zároveň slouží jako jediný unikátní identifikátor objektu. Nelze se totiž spolehnout, že při implementaci dokážeme ohlídat systém zacházení s referencemi. Proto mají objekty přetíženou metodu `Equals()`, která porovnává identifikační čísla objektů. Díky tomu lze porovnávat objekty. Metodu `Equals()` používají kolekce na zjištění přítomnosti prvku v kolekci, vyjmutí prvku z kolekce a další operace. Práce se seznamy obsahující vrcholy nebo tetrahedrony je potom velmi jednoduchá.

Další výhodou tohoto objektového přístupu je možnost zapouzdření objektů. To znamená, že můžeme používat objekty více jako samostatné prvky, ke kterým se vážou metody pracující nad těmito objekty. V předchozí E^2 variantě struktur byl tento přístup problematický, protože objekty byly pasivní a nemohly provádět například operace se sousedy, protože o sobě navzájem nevěděly. Uchovávaly pouze indexy do pole. V novém řešení objekty uchovávají přímo referenci na druhý objekt, takže mohou jednoduše provádět operace jako zjištění protějšího bodu ze sousedního tetrahedronu a podobně. Tento přístup by byl možný i v E^2 variantě, ale vyžadoval by komplikovanější řešení.

Další rozdíl je ten, že program nevyužívá knihovnu na konstrukci Delaunayovy triangulace v E^3, ale načítá již hotovou Delaunayovu triangulaci ze souboru.

Implementace metod v E^3 se ukázala být jako náročná. Implementace velmi zaměstnává prostorovou představivost programátora. Nejhorší fází při implementaci bylo hledání chyb. Kolikrát nebylo možné problém objevit pouze pomocí zobrazovače tetrahedronové sítě a bylo zapotřebí si celou problematiku pouze představovat. Při použití debuggeru toto byl jediný způsob jak zjistit, kde je v postupu chyba. Odladění každé závažnější chyby zabralo několik dní. Nakonec však metody byly úspěšně implementovány.

K vizualizace trojúhelníkových sítí jsem používal prohlížeč napsaný Michalem Varnuškou[24].

[24] Vizualizér napsal Ing. Michal Varnuška PhD. za svého doktorandského studia na Západočeské univerzitě, fakultě aplikovaných věd v Plzni.

8 Experimenty, výsledky

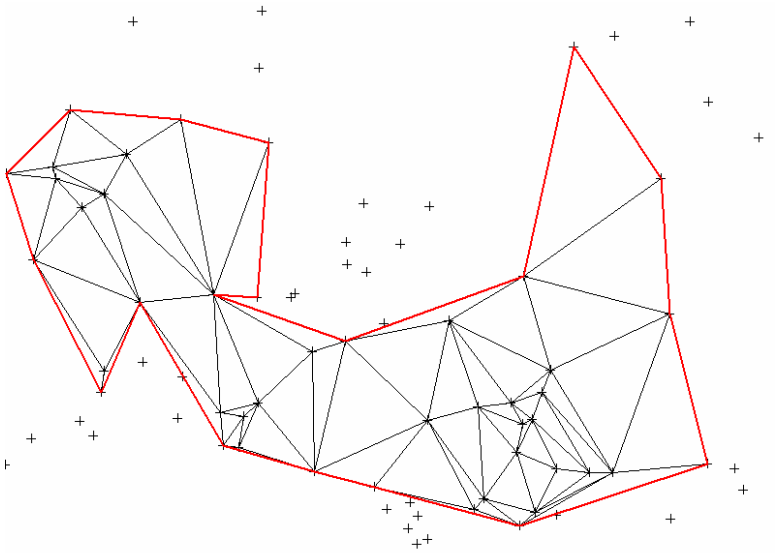
Tato kapitola se věnuje výsledkům této diplomové práce. Nejprve popisuje výsledek implementace metody Maur v prostoru E^2. V další části jsou rozebrány výsledky metod v prostoru E^3. Při posuzování výsledků jsou zvolena následující kritéria.

- Úspěšnost metody. Jedná se o posouzení, zda metoda je funkční, popřípadě pro jaké vstupy funkční je a pro jaké není.
- Numerická stabilita metody. Zde se posuzuje, jak moc je metoda citlivá na numerické nepřesnosti. Toto hledisko je méně podstatné než úspěšnost metody.

Při hodnocení výsledků není posuzována rychlost metody. Není to cílem diplomové práce. Jelikož optimalizace algoritmů je časově náročná, při implementaci byly voleny metody brutální síly. Cílem diplomové práce bylo vyzkoušet metody a jejich stabilitu. Rychlost je až dalším možným posuzovaným hlediskem, kterému se tato diplomová práce z časových důvodů nevěnuje.

8.1 Maur E^2

Metoda Maur v prostoru E^2 je stručně řečeno funkční a použitelná. Metoda dává dobré výsledky. Algoritmus byl schopný ve většině případů vytvořit požadované CDT. Lze říci, že metoda funguje úspěšně pro všechny běžné vstupy. Ukázka výstupu metody je zobrazena na Obr. 8.1.



Obr. 8.1: Ukázka CDT vytvořeného algoritmem Maur. Červené hrany jsou povinné. Výsledné CDT bylo oříznuto na tyto povinné hrany. To je důvod, proč v triangulaci body, které nejsou propojeny hranami.

Při testování metody nikdy nenastal případ, kdy by se algoritmus zasekl. Toto je docela důležitý poznatek. Experimentálně je tedy ověřeno, že při použití regulární triangulace tak, jak je použita v algoritmu Maur, algoritmus lokálního prohazování vede k cíli.

Slabé místo metody je v numerické stabilitě. Největší problémy způsobuje hledání kruhu parametrů paraboloidu. Při vnucování velmi dlouhých hran metoda není schopna nalézt střed kružnice, podle které se vypočítávají rovnice paraboloidů. Problém se objevuje jen ve výjimečných případech. Tento problém by mohl být řešen použitím nějaké speciální

numerické knihovny. Použití speciálních výpočetních metod je v mnoha případech nevyhnutelné.

Při srovnání s programem CDT IK[25] se ukazuje numericky stabilnější program CDT IK. Tento program používá knihovnu ShewLib.dll[26], která provádí orientační testy a testy polohy bodů vůči kružnici. Tato knihovna poskytuje vysokou numerickou stabilitu. Problém numerické nestability by tedy mohl být řešený pomocí lepšího výpočetního modulu.

Kromě numerických problémů se metoda ukázala být funkční. Pokud nenarazila v ojedinělých případech na numerické problémy, produkovala výsledky stejné jako referenční metoda CDT IK.

8.2 CDT E^3

Metody konstrukce CDT v prostoru E^3 byly navrženy tři. Je to metoda Maur E^3, lokální prohazování trojúhelníků a metoda rekonstrukce zasažené oblasti. Rozšíření metody Maur do prostoru E^3 nebylo implementováno. Důvody byly již popsány v kapitole 6.3.

Ostatní dvě metody implementovány byly. Podstatná část výsledků metod je popsána v kapitolách 6.4 a 6.5. V této kapitole budou rozebrány pouze praktické experimenty s metodami.

Metody lokálního prohazování trojúhelníků a metodu rekonstrukce zasažené oblasti pomocí znovuvytvoření se podařilo implementovat. Jak již bylo řečeno v předchozích kapitolách, obě metody se ukázaly být nepoužitelné. Je to jejich nízkou úspěšností. Důvody byly popsány v předchozích kapitolách. Praktické experimenty s metodami tyto důvody potvrzují.

Pro obě metody byly provedeny následující experimenty. Metody byly vyzkoušeny na různých datech: body na povrchu koule, body v mřížce a náhodně rozmístěné body.

Body na povrchu koule

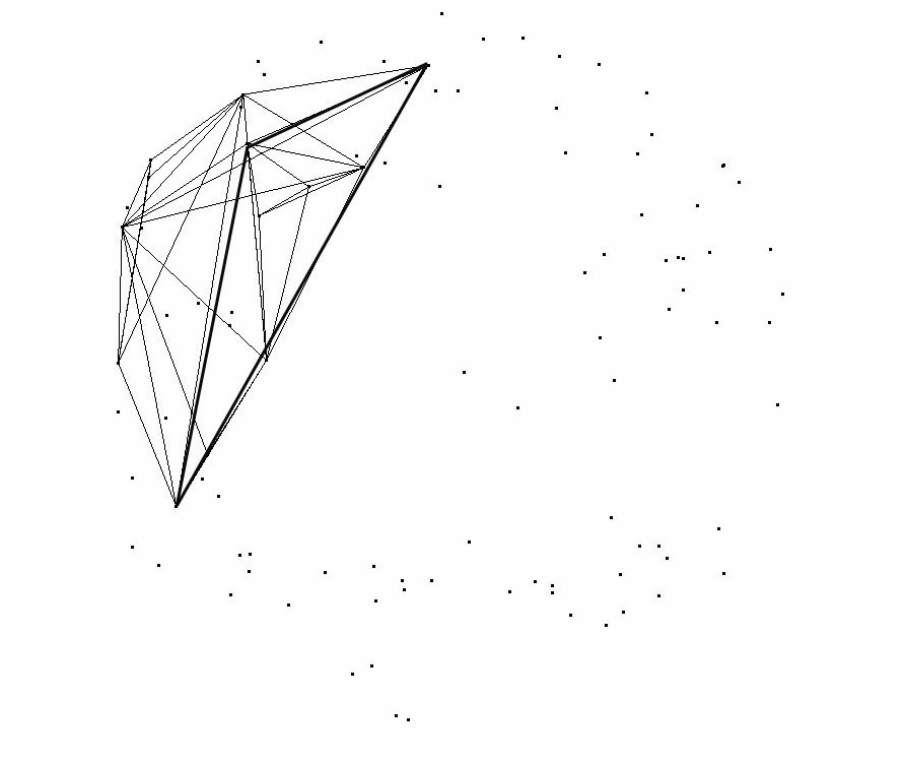
Delaunayova triangulace bodů ležících na povrchu koule se vyznačuje tím, že na povrchu vzniká velké množství tenkých tetrahedronů zatímco uvnitř koule jsou tetrahedrony velké. Pokud vnucujeme trojúhelník takový, jehož body nejsou příliš vzdálené, leží tento trojúhelník leží skoro na povrchu, to znamená, že vnitřek koule protíná jen málo. Takové trojúhelníky metoda vnucuje překvapivě úspěšně. Z naměřených testů vyplývá úspěšnost asi 70% a to dokonce i pro relativně velké zasažené oblasti (okolo 10 až 30 tetrahedronů). Důvod tak vysoké úspěšnosti je právě ve tvaru zasažené oblasti. Zasaženou oblast na povrchu koule znázorňuje Obr. 8.2.

Tato zasažená oblast vznikla vnucením trojúhelníku, který leží na jejím kraji směrem ke středu. Znamená to, že zasažená oblast není rovnoměrně rozmístěná na obě strany. To je tím, že na povrchu koule je mnoho tetrahedronů, které jsou trojúhelníkem protnuty. Ve středu oblasti jsou tetrahedrony organizovány tak, že jejich stěny jsou skoro rovnoběžné s vnuceným trojúhelníkem. Proto jenom málo těchto vnitřních tetrahedronů je protnuto vnuceným trojúhelníkem.

---

[25] Program byl napsán Doc. Dr. Ing. Ivanou Kolingerovou na Fakultě Aplikovaných Věd, Západočeské univerzity v Plzni.

[26] Knihovna je vytvořena prof. J. R. Shewchukem z Kalifornské univerzity v Berkeley.



Obr. 8.2: Zasažená oblast na povrchu koule.

To, že jedna část zasažené oblasti je větší než část druhá ležící na druhé straně trojúhelníku, zapříčiní to, že algoritmus jednodušeji oblast přeskupí. Přesné vysvětlení tohoto tvrzení podat nedokážeme. Algoritmus se z nějakého důvodu proto méně zasekává, když má za vstup oblast splňující výše uvedená kritéria.

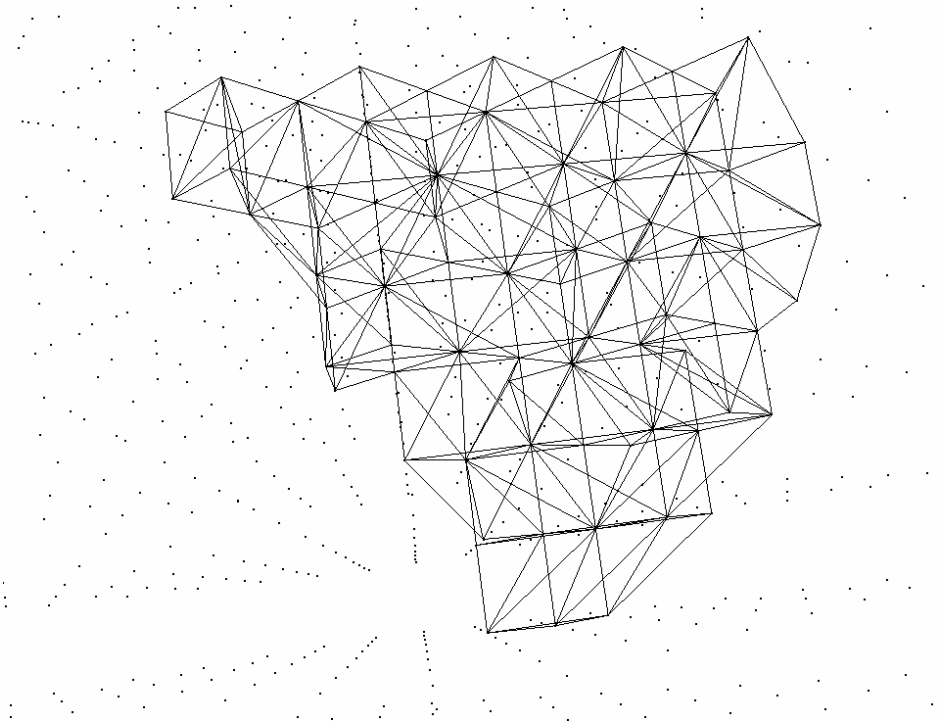
Druhá možnost je vnutit trojúhelník tak, aby protínal kouli skrz její vnitřek. S takovou oblastí měl algoritmus velké problémy. Skoro žádnou takovou oblast se nepodařilo přeměnit na výslednou. Problémem jsou zde pravděpodobně nevyvážené tetrahedrony, které jsou u povrchu koule malé a je jich mnoho, které jsou protnuty vnuceným trojúhelníkem. Naopak uvnitř koule je tetrahedronů protnutých málo a tyto tetrahedrony jsou větší. Toto je pravděpodobně zdrojem problému.

Body v mřížce

Bohužel tyto body nebylo možné plně otestovat. Důvod je ten, že implementovaná metoda je citlivá na singulární případy. Delaunayova triangulace vzniklá z bodů tvořících mřížku je velkým problémem vzhledem k singulárním případům. Body v mřížce jsou obecně problematickým vstupem pro velké množství algoritmů.

Řešení singulárních případů nebylo implementováno, protože zabírá při implementaci hodně času. Například u implementace algoritmu Maur jenom třetina času byla věnována řešení singulárních případů. Přesto není stále algoritmus Maur odolný na všechny numerické chyby. Nedává velký smysl vylepšovat ani jeden z algoritmů metod v E^3 na odolnost proti singulárním případům, protože metody zatím mají problémy jiné, závažnější.

Příklad jednoho řešení singulárního problému je následující. Pokud nějaký bod leží na vnuceném trojúhelníku, je potřeba trojúhelník rozdělit na tři menší. Toto platí, pokud bod leží uvnitř. Pokud leží na hraně, dělí se trojúhelník na dva. Bod ležící na trojúhelníku byl častým výskytem právě v případě experimentů s daty, kde body leží na mřížce. S těmito daty měly problémy oba algoritmy. Ukázka zasažené oblasti, kterou algoritmus nalezl je na Obr. 8.3.



Obr. 8.3: Body ležící v mřížce. Na obrázku je zasažená oblast.

Náhodně rozmístěné body

Tato bodová množina se ukázala být nejvhodnější pro hlubší experimenty. Vhodná je proto, že se nejedná o žádný extrémní případ, kde by se algoritmus choval výjimečně. Na této množině bodů bylo vyzkoušeno asi třicet možností vnucených trojúhelníků. Povinné trojúhelníky se lišily ve velikosti zasažené oblasti, kterou vytvořily. V některých případech algoritmus selhal jindy byl schopen trojúhelník vnutit. Testovaly se oba algoritmy. U algoritmu lokálního prohazování trojúhelníků se testovala varianta popsaná v kapitole 6.4, kde jsou trojúhelníky ze seznamu vybírány náhodně. Náhodné vybírání trojúhelníků zapříčiňuje, že algoritmus jednou konverguje a při opětovném spuštění ne. Výhoda je ta, že při určitém počtu opakování víme, zda metoda pravděpodobně nemůže pro daný vstup konvergovat nikdy nebo zda je možnost, že při určité sekvenci swapů může algoritmus dojít k cíli. Rozdíl je pro naše úvahy celkem podstatný, jak již bylo popsáno v předchozích kapitolách.

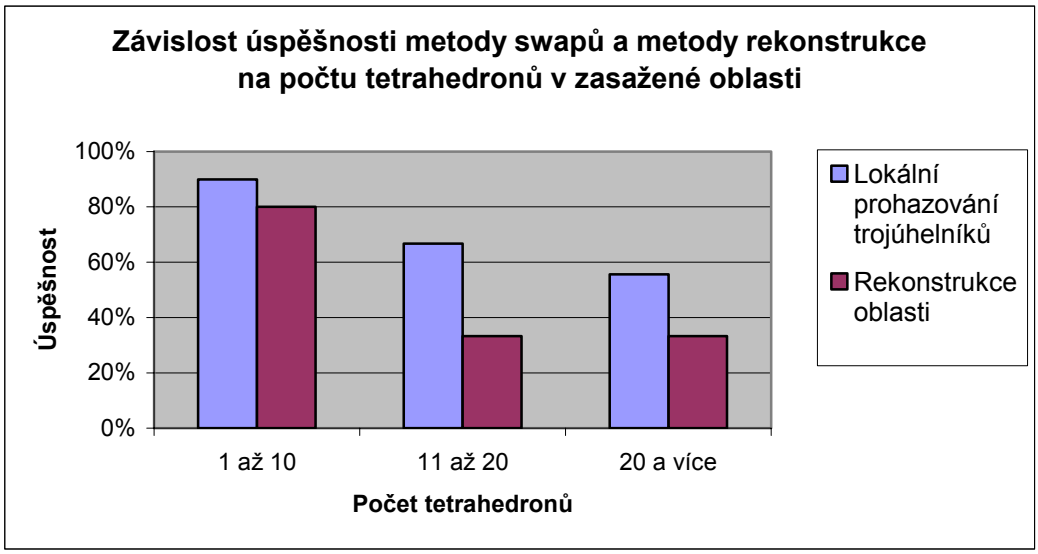
Metoda rekonstrukce oblasti pomocí znovuvytvoření náhodné rozhodování neobsahuje, proto byla spuštěna pouze jednou. Metoda prohazování trojúhelníků byla spouštěna celkem desetkrát. Výsledky zjištěné experimenty jsou uvedeny v následující tabulce. Počet TH určuje kolik tetrahedronů obsahuje zasažená oblast. Pokud se metoda pro daný vstup dostala k cíli aspoň jednou z deseti spuštění, je příslušné políčko označeno písmenem A. Pro metodu rekonstrukce oblasti úspěch znamená to, že byla schopná obnovit síť tak, aby v triangulaci nevznikaly díry.

tabulka 1: Úspěšnost metody prohazování tetrahedronů. Písmenko A značí úspěch metody na daný počet tetrahedronů v zasažené oblasti.

Počet TH	2	2	2	3	4	4	4	5	9	10	12	15	15	16
Prohazování	A	A	A	A	A		A	A	A	A		A	A	A
Rekonstrukce	A	A	A	A	A		A	A		A		A	A	

Počet TH	16	19	20	20	20	21	21	22	23	24	31	33	37	82
Prohazování	A	A	A			A		A			A	A	A	
Rekonstrukce		A									A	A	A	

Následující graf znázorňuje závislost úspěšnosti metod na velikosti zasažené oblasti.

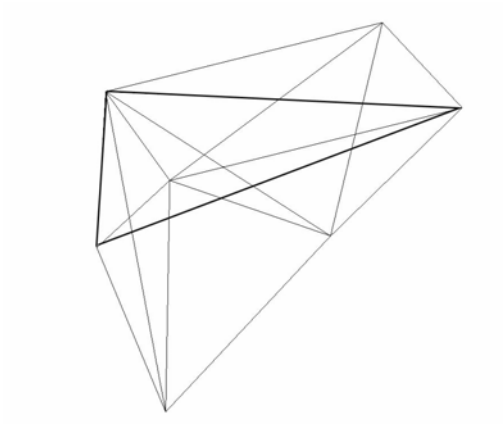


Graf 1: Závislost úspěšnosti metod na velikosti zasažené oblasti. Porovnány jsou dvě metody: lokální prohazování trojúhelníků a rekonstrukce zasažené oblasti znovuvytvořením

Zde je zapotřebí říci, že nejsou důležitá přesná čísla procentuelních úspěšností, protože nejsou podložena dostatečným počtem experimentů. Výsledky jsou statisticky věrohodné pouze pokud jsou zaokrouhleny zhruba na desítky procent. Z takovýchto výsledků lze vyvozovat závěry. Přesnější výsledky však nepotřebujeme, jelikož není potřeba porovnávat metody s metodami jiných autorů.

Při pohledu na graf je zřejmé, že metoda je nejúspěšnější pro malé oblasti. Přesto se v malých oblastech objevily případy, v kterých nebylo možné trojúhelník vnutit. V těchto případech dle mého názoru ani nemohl trojúhelník být vnucen z důvodu tvaru oblasti a rozložení bodů. Toto potvrzuje teorii, která byla popsána v předchozích kapitolách. Existují případy, v kterých neexistuje CDT. Také existují případy bodů a povinných trojúhelníků, v kterých neexistuje jakákoliv tetrahedronizace oblasti. Tímto případem je například tzv. Schöhardtův polyhedron (viz kapitola 6.1.1). Problém se řeší přidáním vhodného bodu.

Obr. 8.4 znázorňuje výše zmíněnou oblast, kterou se nepodařilo při žádném pokusu tetrahedronizovat. Na tomto případě selhaly oba algoritmy. Jedná se pravděpodobně o situaci, kde není možné trojúhelník vnutit. Problém by mohl být řešitelný metodou přidání bodu, která je popsána v [shew02a].

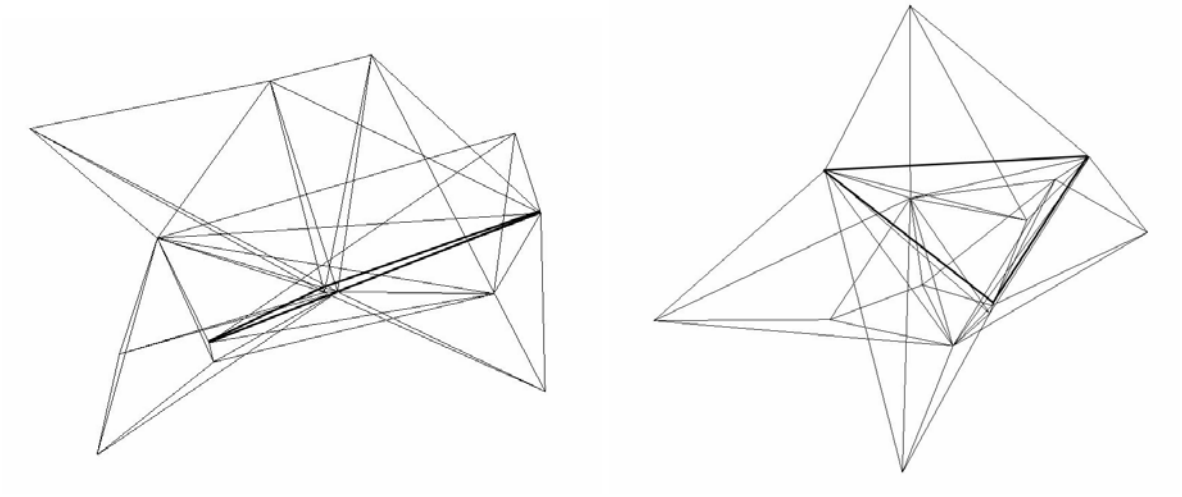


Obr. 8.4: Zasažená oblast, kterou není možné tetrahedronizovat. Oblast obsahuje 4 tetrahedrony.

Jak experimenty ukázaly, nezáleží mnoho na vnějším tvaru oblasti. Intuitivně bychom řekli, že algoritmus se chová lépe, pokud zasažená oblast neobsahuje žádné ostré výběžky a její tvar je spíše konvexní. Experimenty ukázaly, že tomu tak není. Jeden takový příklad je na Obr. 8.5. Zde je nekonvexní oblast s ostrými výběžky tetrahedronů. Do této oblasti se podařilo vnutit povinný trojúhelník pomocí metody lokálního prohazování. Metoda došla k cíli v osmi z deseti spuštění. Metoda rekonstrukce zasažené oblasti zde selhala.

Některé jiné oblasti, které byly hezky konvexní se nepodařil změnit na oblast cílovou ani jednou z deseti spuštění. Zde se ukázalo, že na tvaru vnější hranice metoda lokálního prohazování trojúhelníků příliš citlivá není.

Experimentů bylo provedeno mnoho. Všechny tyto pokusy lze najít na přiloženém CD, kde je i prohlížeč tetrahedronových sítí.



a) b)

Obr. 8.5: Nekonvexní zasažená oblast obsahující výběžky tetrahedronů. Na obrázku a) je pohled z boku, povinný trojúhelník leží vodorovně a skoro splývá s osou pohledu. Na obrázku b) je pohled z vrchu.

Při použití lokálního prohazování trojúhelníků některé trojúhelníky bylo možné vnutit jenom někdy, jiné algoritmus nevnutil ani při dvacátém opakování. Výsledky experimentů jsou takové, že případů, kdy se nepodařilo trojúhelník vnutit při žádném opakování, je 20%. Případů, kdy se trojúhelníky vnutily vždy, je 30%. Zbytek jsou případy, kdy někdy algoritmus trojúhelník vnutil a jindy ne. Těchto případů je 50%.

Všechny tyto výsledky jsou velmi užitečné pro další možný postup řešení daného problému. Nejprve se podívejme na závislost úspěšnosti algoritmu na velikosti zasažené oblasti. Zde je

vidět, že obě metody jsou úspěšnější pro menší zasažené oblasti. Metoda rekonstrukce oblasti je o něco méně citlivá na velikost oblasti, ale je méně úspěšná než metoda lokálního prohazování trojúhelníků. Metoda lokálního prohazování trojúhelníků se ukázala být velmi úspěšná pro malé oblasti. Případy, kde metoda selhala, se ukázaly být jako případy, které nemají řešení pomocí žádného algoritmu, který provádí pouze sekvenci swapů. S velkou pravděpodobností se jednalo o případy, kde řešení neexistuje vůbec. Zde by bylo potřeba provést přidání bodu do zasažené oblasti. Pro velké oblasti se metoda ukázala být úspěšná méně a úspěchu bylo dosaženo až po několikanásobném opakování algoritmu. Řešením by zde bylo rozdělit povinný trojúhelník na tři menší trojúhelníky a použít algoritmus pro tři části zvlášť. Tím by se zvýšila celková pravděpodobnost úspěchu metody.

Výše popsaných 50% případů u metody lokálního prohazování trojúhelníků je těch, kdy metoda potřebovala několik opakování, aby našla správnou posloupnost swapů. Toto byl případ i pro malé oblasti. Řešení tohoto problému může být takové, že se ponechá metoda tak, jak je. Metoda bude prováděna tak dlouho dokud trojúhelník do oblasti nevnutí. Zde by bylo rozumné vložit nějaké omezení maximálního počtu opakování. Druhé řešení je použít prohledávání grafu swapů, které bylo popsáno v kapitole 6.4. Tyto dvě řešení sice k cíli mohou vést, ale jsou poměrně časově náročné a nemusí stále zaručit stoprocentní výsledek. Elegantnějším řešením by bylo použít nějaký postup, jako postup v kapitole 6.1.1. Zde se používá prioritní fronta. Jak by ovšem tato fronta vypadala pro námi navrženou metodu, nedokážeme nyní popsat. Implementace fronty nebo prohledávání grafu by byla časově velmi náročná.

Jak bylo řečeno, 20% nemá řešení pomocí běžného algoritmu prohazování a nebo nemá řešení vůbec. Přidání bodu tento problém řeší. Způsob, jakým má být bod přidán je popsán v [shew02a]. Takovéto rozšíření algoritmu je bezpodmínečně nutné, protože jinak by pro tyto případy neexistovalo řešení žádné. Přidání bodu je složitý postup, který je implementačně náročný. Toto rozšíření překračuje možnosti diplomové práce. To je jeden z důvodů, proč výsledkem diplomové práce nemohla být fungující metoda pro CDT v prostoru E^3. Obecně je časová náročnost na implementaci podobných algoritmů velmi vysoká.

Těmito návrhy na vylepšení bychom získali vnucenou hranu. Bohužel bychom stále neměli CDT. K tomu by bylo zapotřebí provést druhou část algoritmu lokálního prohazování trojúhelníků, a to obnovení Delaunayova kritéria v oddělených oblastech.

Další překážkou pro algoritmus by mohlo být vnucení více trojúhelníků, které leží v těsné blízkosti. CDT by pro daný případ nemuselo opět existovat. I tento problém se řeší vhodným přidáním bodu do triangulace popsaným v [shew02a]. Zde je popsáno, jak přidávat body tak, aby bylo možné vnutit trojúhelníky, které sdílí stejný vrchol a je mezi nimi malý úhel.

8.3 Shrnutí výsledků

Algoritmus Maur v prostoru E^2 se podařilo úspěšně implementovat. Algoritmus je schopen vytvářet CDT ze vstupní Delaunayovy triangulace. Algoritmus byl otestován na vstupních datech a ve většině případů je plně funkční. V některých případech selhal. Důvodem jsou však numerické chyby při výpočtech. Návrh, jak by tyto problémy mohly být řešeny je popsán v kapitole 8.1. Na jejich odstranění bohužel není v rámci diplomové práce čas. Algoritmus nejenže dokáže konstruovat CDT, ale je i dobrým teoretickým zázemím pro výzkum metod v prostoru E^3. Algoritmus totiž představuje použití nestandardního přístupu přechodu do vyšší dimenze [27] za použití regulární triangulace. Důkaz užitečnosti těchto znalostí je ten, že jeden

[27] Přechod do vyšší dimenze je postup konstrukce Delaunayovy triangulace popsaný v kapitole 2.3.3.

z mála známých algoritmů na konstrukci CDT je založen na podobných principech. Je jím inkrementální algoritmus popsaný v kapitole 6.1.1. Zde je potřeba uvést, že algoritmus Maur byl navrhován v době, kdy tento inkrementální algoritmus ještě nebyl zveřejněn.

Další postup diplomové práce spočíval v navržení postupu použití metody Maur na komprese trojúhelníkových sítí. Problematika byla nastudována a závěr je, že není možné algoritmus efektivně použít. Vše bylo zdokumentováno v kapitole 4.

Rozšíření metod do prostoru E^3 se podařilo implementovat. Bohužel tyto metody nebyly schopné spolehlivě konstruovat CDT. Problém není v chybě v implementaci ale v principiálních problémech metod. To znamená, že metody potřebují být doplněny o další vylepšení, která ovšem zdaleka převyšují rozsah diplomové práce. Odhadovaný čas na implementaci vylepšeních je jeden rok. Metody však přinesly zisk mnoha znalostí problematiky CDT v E^3. Tyto znalosti byly potvrzeny experimenty a jsou velmi užitečné právě pro možný budoucí vývoj a vylepšení metod.

Metoda rekonstrukce oblasti byla úspěšná asi v polovině testovaných případů. Metoda se ukázala být spolehlivější na malé oblasti. To, že metoda nebyla úspěšná znamená, že v rekonstruované zasažené oblasti vznikaly díry. Toto je však závažný problém. Důvodem tohoto problému jsou většinou numerické chyby, na které je metoda náchylná. Existovaly i případy, kde selhání metody nebylo dáno numerickými chybami, ale tím, že pro daný vstup neexistoval výstup. Zhodnocení metody je následující. Metodu je doporučeno nepoužívat a dále nerozvíjet, protože je založena na principech, které jsou náchylné na chyby. Metoda však posloužila pro vytvoření teoretických závěrů, které mohou být použity pro vylepšení metody lokálního prohazování trojúhelníků.

Metoda lokálního prohazování trojúhelníků se ukázala být použitelnější než metoda rekonstrukce oblasti. Přesto však nebyla schopná vždy vnutit povinný trojúhelník. Algoritmus se v mnoha případech zasekl. Důvodem opět není chyba při implementaci, ale principielní problém celé metody. Důvody pro neúspěch metody jsou tři. Jeden je, že neexistuje řešení vstupního problému žádné. V takových případech se do triangulace přidávají body. Takto se řeší i druhý problém a to, že neexistuje posloupnost swapů, která by trojúhelník vnutila. Třetím problémem je to, že metoda sice došla k cíli, ale jen v procentuelně nízkém počtu případů. V ostatních případech se algoritmus zasekl. Toto je již problém detailního návrhu metody, který může být odstraněn například použitím prioritní fronty. Tato fronta by řídila pořadí swapů tetrahedronů. Samotný návrh fronty však není znám a objevení takového mechanismu není snadnou záležitostí.

9 Závěr

V rámci diplomové práce byl vytvořen detailní návrh metody Maur pro prostor E^2. Tato metoda byla úspěšně implementována a ukázala se být funkční a použitelná v prostoru E^2. Metoda sice nekonkuruje v rychlosti ostatním existujícím metodám. Cílem diplomové práce nebylo testovat rychlost metod ani vytvářet optimalizované rychlé metody. I přes možnou optimalizaci by metoda nebyla rychlejší než většina jiných metod. Je to tím, že používá poměrně složité výpočty pro svou konstrukci. Přínos algoritmus Maur E^2 je ten, že představuje použití nestandardního přístupu v konstrukci CDT pomocí přechodu do vyšší dimenze.

Byla prozkoumána problematika použití Delaunayovy triangulace při kompresích trojúhelníkových sítí. Jedním z bodů diplomové práce bylo zjistit, zda je možné použít algoritmus Maur při kompresích. Metoda se ukázala být pro toto použití nevhodná. Důvody byly detailně popsány.

Dalším cílem diplomové práce bylo vyzkoušet rozšíření metody Maur do prostoru E^3. To se však při analýze se zkušenostmi z E^2 varianty ukázalo být nemožné. Proto bylo vyzkoušeno rozšíření dvou jiných metod do prostoru E^3.

První metodou bylo lokální prohazování trojúhelníků. Metoda byla navržena pro prostor E^3 a implementována. Algoritmus konvergoval k cíli jen pro nějaké vstupy. Metoda byla úspěšná pro menší oblasti. Podařilo se zjistit důvod a byl popsán ve výsledcích metody. Přes to, že metoda nefunguje, její vyzkoušení přináší cenné výsledky pro další možné experimenty s CDT v prostoru E^3. Další metodou byla metoda rekonstrukce zasažené oblasti pomocí znovuvytvoření. Tato metoda vždy vnutila do triangulace požadovaný trojúhelník. Na druhou stranu v mnoha případech vznikaly v triangulaci díry. Jedná se o zásadní problém. Příčinu se sice nepodařilo a jistotou určit. Jedná se pravděpodobně o numerické chyby, na které je metoda velmi citlivá. Metoda po zkušenostech byla posouzena jako nevhodná pro konstrukci CDT.

I když žádná z metod nebyla úspěšná, podařilo se zjistit a popsat jejich chování a pokud to bylo možné, tak i zdroj problémů. Vše bylo zdokumentováno v diplomové práci.

Další možný postup ve zkoumání metod by mohl být rozšíření metody lokálního prohazování trojúhelníků o dělení vnuceného trojúhelníku pomocí vkládání pomocných bodů do triangulace. Výsledkem by sice nebylo CDT vytvořené nad původní množinou bodů, ale zvýšila by se pravděpodobnost úspěchu metody. Další možností by bylo zavedení nějakého postupu, který bude řídit chod prohazování trojúhelníku tak, aby se algoritmus nemohl dostat do slepé uličky.

Ačkoliv implementované metody nejsou plně funkční, byla naplněna podstata diplomové práce a to popsat chování metod v prostoru E^3. To se dle mého názoru úspěšně podařilo. Diplomová práce mi byla jako celek přínosem v získání rozsáhlých znalostí a praktických zkušeností z problematiky Delaunayovy triangulace.

Závěrem bych chtěl poděkovat dvěma lidem za pomoc s diplomovou prací. Je to jednak Doc. Dr. Ing. Ivana Kolingerová, která byla mým vedoucím pro diplomovou práci. Druhý člověk je Ing. Pavel Maur, který mi pomáhal mi s promýšlením možného rozšíření metody do prostoru E^3.

Použitá literatura

[angla97a] ANGLADA, M. V. An improved incremental algorithm for constructing Restricted Delaunay triangulation, *Comput. & Graphics*, 21st edition, 1997. s. 215 – 223.

[edel96a] EDELSBRUNNER. H, SHAH N. R. Incremental topological flipping works for regular triangulations. In *Algorithmica*. 15th edition. 1996, s. 223-241.

[fac95a] FACELLO, M. A. Implementation of a randomized algorithm for Delaunay and regular triangulations in three dimensions. In *Computer Aided Geometric Design*. 12th edition., 1995. s. 349-370.

[joe89a] JOE, B. Construction of three-dimensional Delaunay triangulations using local transformations. In *SIAM J. on Scien. Statist. Computing,* 10th edition., 1989. s. 718-741.

[kim99a] KIM, Yang-Soo, et al. Geometric compression using Delaunay triangulation. *Seventh Pacific Conference on Computer Graphics and Applications*. 1999, s. 118.

[koh02a] KOHOUT, J. Paralelní Delaunayova triangulace ve 2D a 3D. *Diplomová práce, Západočeská univerzita v Plzni, fakulta aplikovaných věd, katedra informatiky a výpočetní techniky*. 2002.

[maur04] MAUR P., KOLINGEROVÁ I.: The Employment of regular triangulation for constrained Delaunay triangulation, In *workshop on Computational Geometry and Applications, ICCSA Konference, Assisi*. 2004, s. 198-207.

[plucker] http://www.loria.fr/~lazard//ARC-Visi3D/Pant-project/files/plucker.html

[shew00a] SHEWCHUK, R. J. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Annual Symposium on Computational Geometry*. 16th edition, 2000. s. 350-359. ISBN 1-58113-224-7.

[shew02a] SHEWCHUK, R. J. Constrained Delaunay tetrahedralizations and provably good boundary recovery. In *11th International Meshing Roundtable, Sandia National Laboratories*, 11th edition, 2002, s. 193-204.

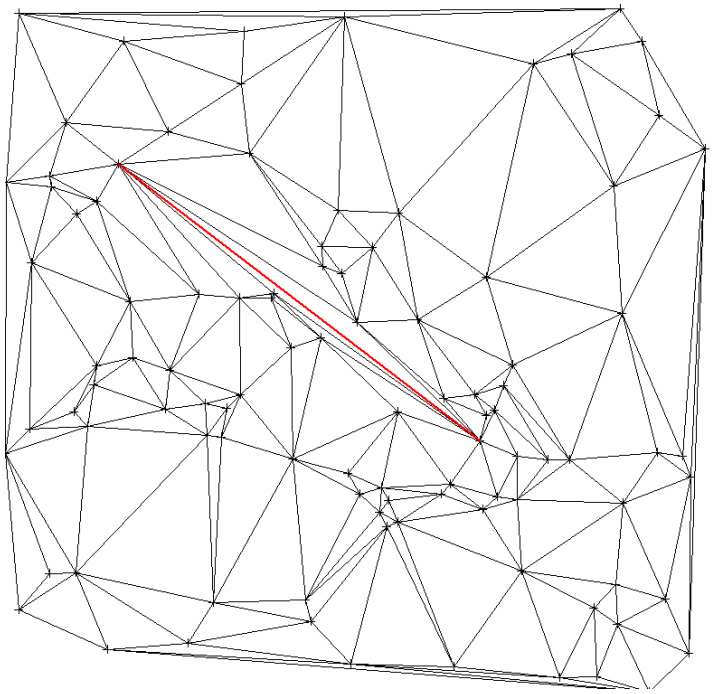
[shew03a] SHEWCHUK, R. J. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *Annual Symposium on Computational Geometry*. 19th edition, 2003. s. 181-190. ISBN 1-58113-663-3.

[sloan93a] SLOAN, S. W. A fast algorithm for generating constrained Delaunay triangulations. In *Computers & Structures*. 47th edition., 1993. s. 441-450.

Příloha A

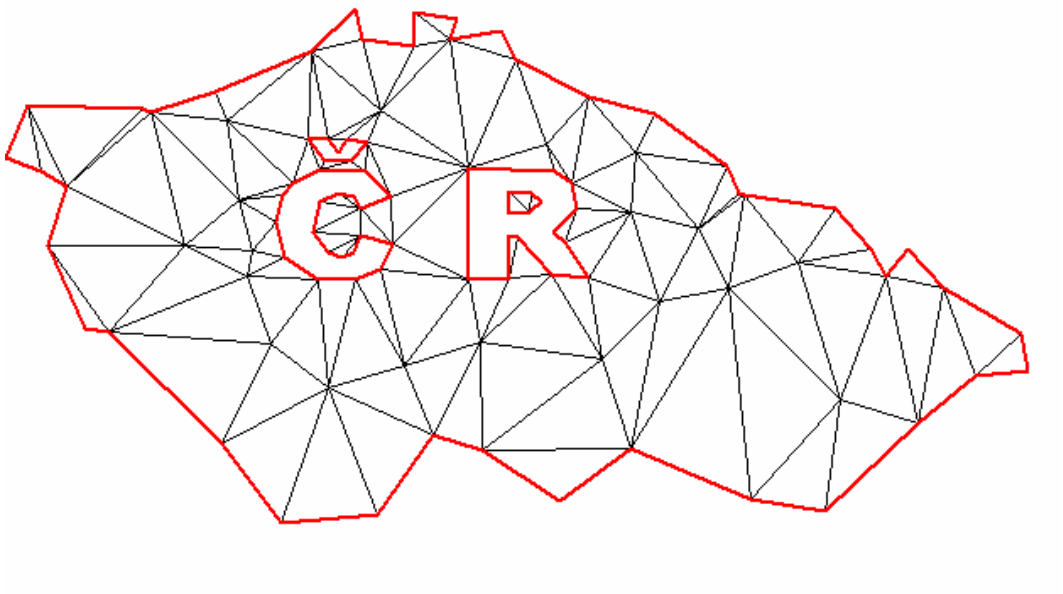
Výsledky CDT E²

Tato příloha obsahuje výstupy z CDT E^2 programu. Obr. A.1 zobrazuje vnucenou hranu do triangulace. Hrana je zvýrazněna červeně.



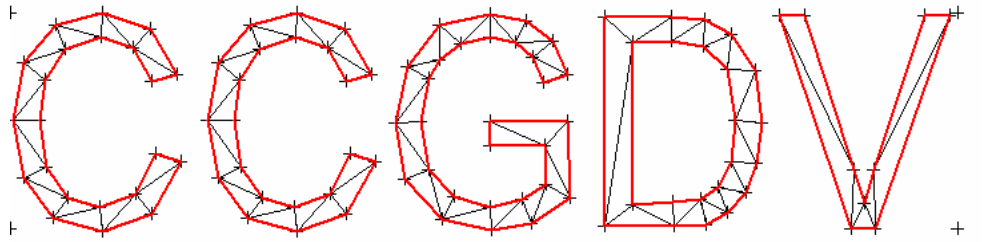
Obr. A.1: Ukázka CDT oříznutého na povinné hrany.

Obr. A.2 ukazuje výstup s vnucenými hranami (červeně) a oříznutý na tyto hrany. Ořezávání bylo zvoleno pro vnější oblast. Vnitřek písmen ČR je také vyříznutý, protože algoritmus ořezávání prošel podruhé přes povinnou hranu.

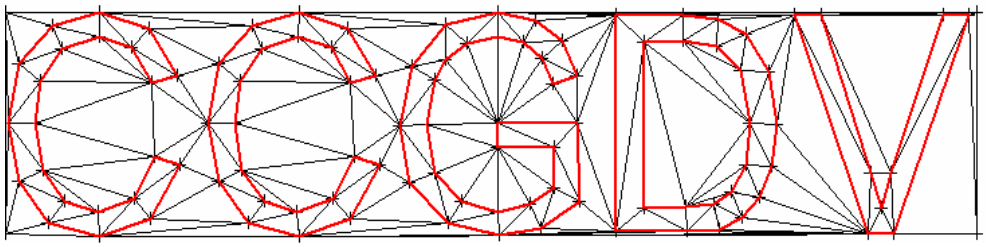


Obr. A.2: Ukázka CDT oříznutého na povinné hrany.

Na Obr. A.3 je další ukázka výstupu programu. Zde je oříznuta vnější oblast. Neoříznutá varianta je na Obr. A.4.

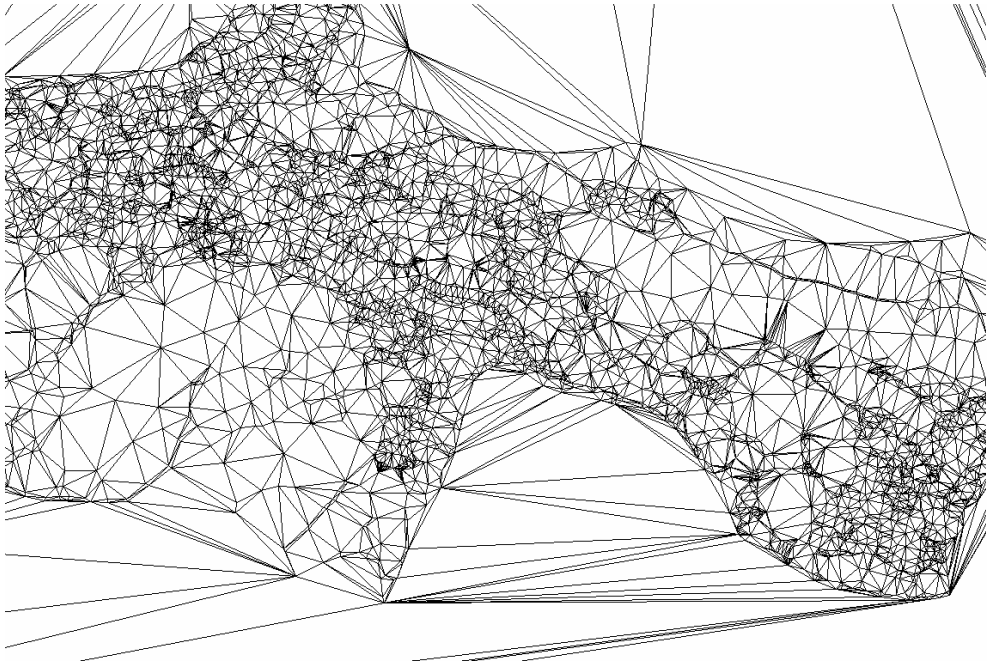


Obr. A.3: Ukázka CDT oříznutého na povinné hrany



Obr. A.4: Ukázka CDT, které nebylo ořízнuto.

Na Obr. A.5 je znázorněn malý výsek z datové množiny, pro kterou program nedokáže sestrojit CDT. Důvodem jsou numerické chyby, které se při výpočtu objevují.

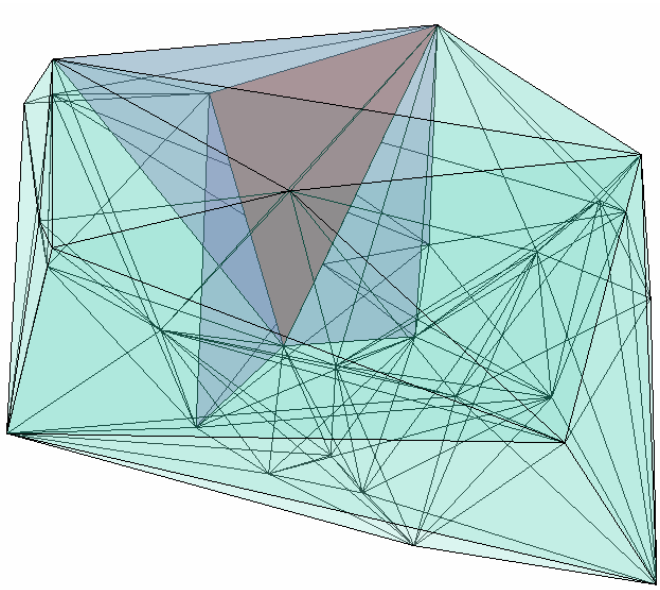


Obr. A.5: Část datové množiny, pro kterou program nedokáže sestrojit CDT z důvodu numerických nepřesností.

Příloha B

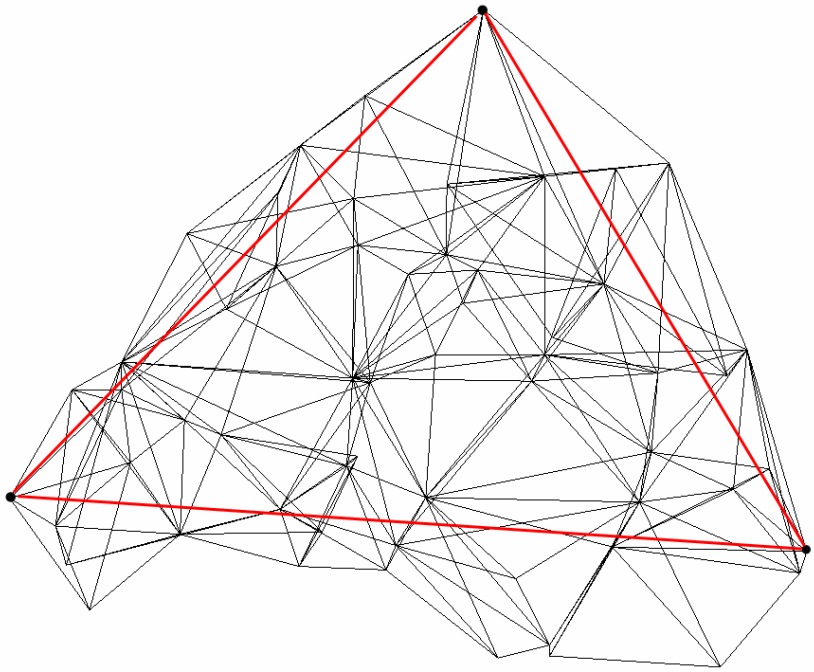
E^2 výsledky

Vstupem pro metody v E^3 byla Delaunayova triangulace v E^3. Jedna taková triangulace je znázorněna na Obr. B.1.

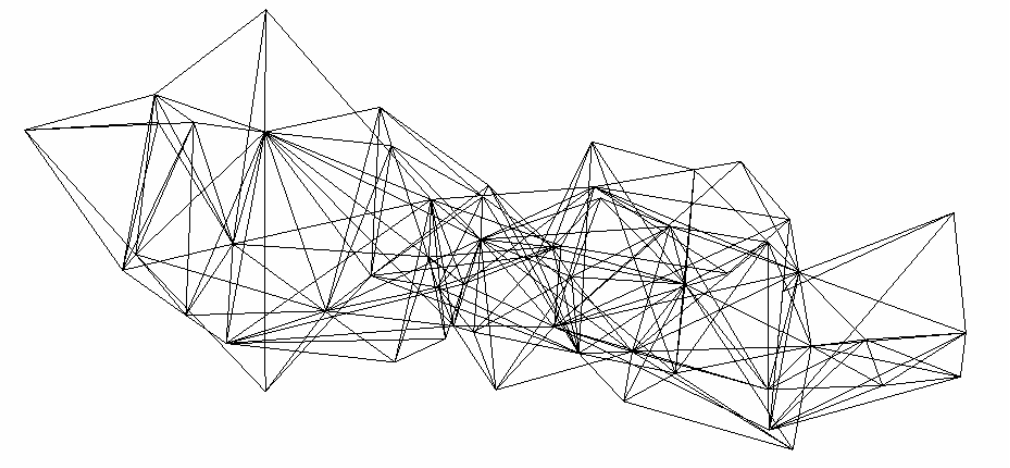


Obr B.1: E^3 triangulace vytvořená nad 30 body náhodně rozmístěnými v prostoru E^3. Tmavší barvou je zvýrazněn jeden tetrahedron v síti. Kolem něj jsou vyznačeny 4 tetrahedrony, které s ním sousedí.

Na Obr. B.2 je znázorněna zasažená oblast při pohledu z vrchu. Na Obr. B.3 je tato oblast při pohledu z boku.

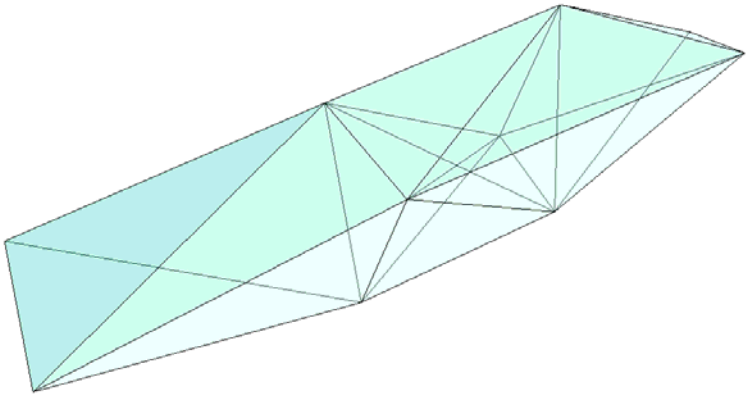


Obr B.2: Zasažená oblast. Tlustými čarami je zde vyznačen vnucený trojúhelník. Zasažená oblast byla nalezena metodou hrubé síly.

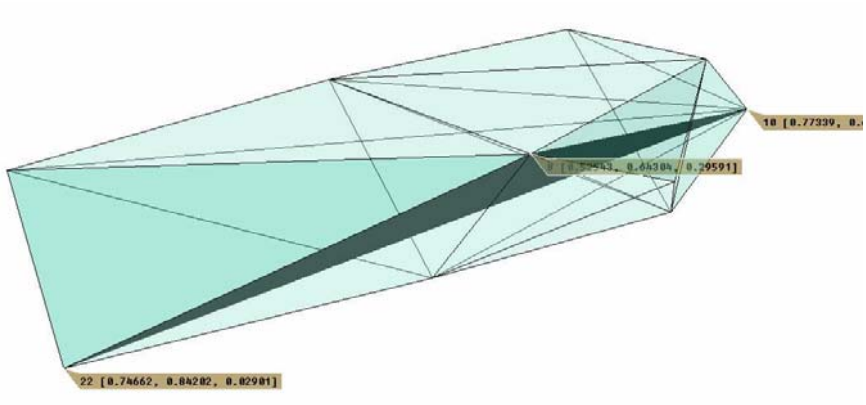


Obr. B.3: Zasažená oblast, pohled z boku.

Na Obr. B.4 je zasažená oblast, do které je vnucen trojúhelník pomocí metody swapů. Oblast obsahující tento trojúhelník je na Obr. B.5.

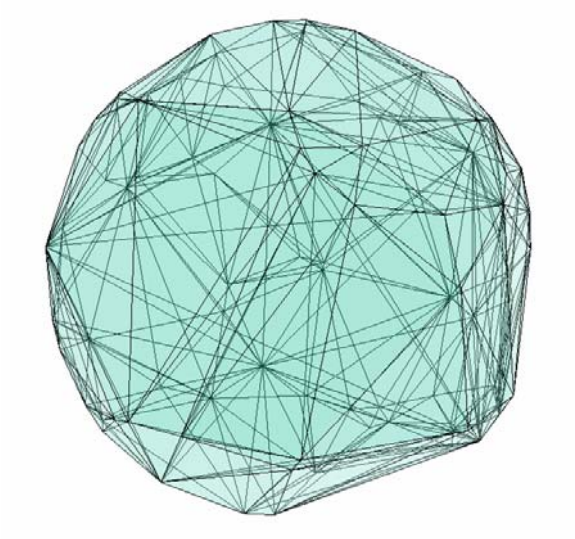


Obr. B.4: Zasažená oblast obsahující 10 tetrahedronů.

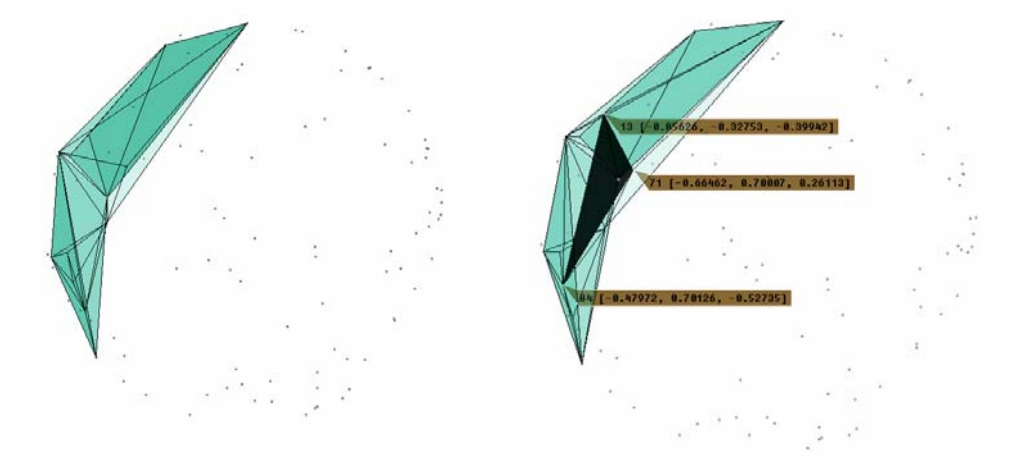


Obr. B.5: Zasažená oblast z Obr. B.4, do které byl vnucen trojúhelník.

Na Obr. B.6 je znázorněna Delaunayova triangulace bodů ležících na povrchu koule. Tetrahedrony na povrchu jsou ploché jak znázorňuje Obr. B.7. Na tomto obrázku je zasažená oblast (vlevo) a do ní vnucený trojúhelník (vpravo).



Obr. B.6: Delaunayova triangulace nad body ležícími na kouli



Obr. B.7: Na obrázku vlevo je zasažená oblast na povrchu koule, na obrázku vpravo je do ní vnucen trojúhelník pomocí metody swapů.

Příloha C

Uživatelská příručka

CDT E^2

Program konstruuje CDT nad vstupní množinou bodů. Body jsou nadefinovány ve vstupním souboru s příponou .dat.

Vstupní soubor

Ukázka vstupního souboru je například tato:

```
5
0.01 0.32
0.15 0.63
0.25 0.25
1.50 0.48
1.22 0.88
2
0 2
2 4
```

V souboru jsou nejprve zadány body a poté povinné hrany. Každé části předchází číslo určující počet položek. V tomto případě je zadaných 5 bodů a 2 povinné hrany. Body jsou zadávány po řádcích. Jejich souřadnice jsou uloženy v pořadí x, y. Hrany definují indexy dvou bodů, které hrana spojuje. Body jsou indexovány od nuly. V souboru jsou tedy dvě hrany: hrana mezi body o souřadnicích [0.01, 0.32], [0.25, 0.25] a hrana [0.25, 0.25], [1.22, 0.88].

Výstupní soubor

Pro výše uvedený vstup je výstupní soubor následující.

```
5
0.01 0.32
0.15 0.63
0.25 0.25
1.5 0.48
1.22 0.88
3
3 4 2
4 1 2
1 0 2
3
-2 1 1
1 -2 1
-2 1 1
```

Nejprve jsou ve výstupu uloženy body stejným způsobem jako vstupní body. Poté jsou uloženy trojúhelníky a po nich indikace povinných hran v trojúhelnících.

Následuje detailní popis položek ve výstupním souboru:

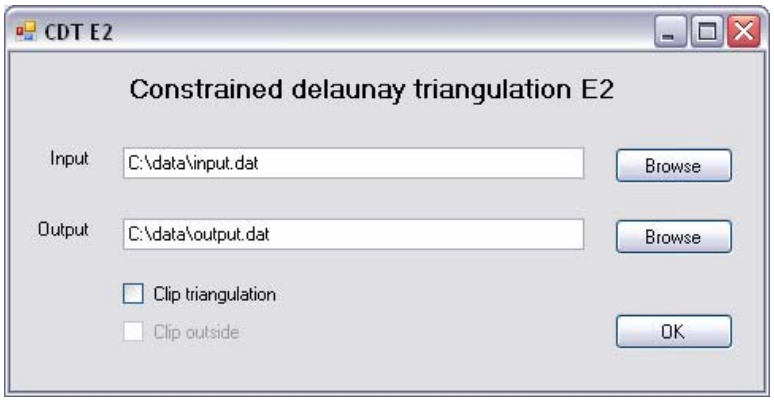
```
<počet bodů N>
<bod 0 X> <bod 0 Y>
<bod 1 X> <bod 1 Y>
…
<bod N-1 X> <bod N-1 Y>
<počet trojúhelníků T>
<troj 0 vrchol 0> <troj 0 vrchol 1> <troj 0 vrchol 2>
<troj 1 vrchol 0> <troj 1 vrchol 1> <troj 1 vrchol 2>
```

```
…
<troj T-1 vrchol 0> <troj T-1 vrchol 1> <troj T-1 vrchol 2>
<počet trojúhelníků T>
<hrana troj 0 vrchol 0> <hrana troj 0 vrchol 1> <hrana troj 0 vrchol 2>
<hrana troj 1 vrchol 0> <hrana troj 1 vrchol 1> <hrana troj 1 vrchol 2>
…
<hrana troj T-1 vrchol 0> <hrana troj T-1 vrchol 1> <hrana troj T-1 vrchol 2>
```

Uložení bodů bylo popsáno výše. Trojúhelníky jsou zadány třemi indexy vrcholů. Indexy začínají od nuly. Indikace povinných hran určuje pro každý trojúhelník, která z jeho hran je povinná. Ve stejném pořadí, jako je pořadí trojúhelníků, jsou nadefinovány trojúhelníky obsahující indikace. Každý trojúhelník má definováno pro každou jeho hranu, zda hrana je povinná. Pořadí hran odpovídá pořadí vrcholů v trojúhelnících. Ke každému vrcholu se váže jeho protější hrana. Povinná hrana je indikována pomocí čísla -2 a nepovinná pomocí čísla 1.

V příkladě výstupního souboru má tedy první trojúhelník jako vrcholy body 3, 4 a 2. Hrany v tomto trojúhelníku jsou definovány v pořadí 42, 32 a 34. Indikace povinných hran jsou -2 1 1. Hrana 42 je tedy povinná. Ve vstupním souboru je tato hrana uložena jako druhá v pořadí. Pořadí uložení hran nijak neovlivňuje výsledek.

Program se spouští souborem CDTE2.EXE. Po spuštění se zobrazí formulář, v němž se nadefinuje jméno vstupního a výstupního souboru i s cestou. Checkbox *Clip triangulation* nastavuje, zda má být triangulace oříznuta. Pokud je zaškrtnut, objeví se další volba *Clip outside*, která určuje, zda má být oříznut vnějšek oblasti nebo vnitřek. K tomu, aby triangulace mohla být oříznuta, musí povinné hrany tvořit v grafu uzavřenou kružnici. Ukázka formuláře je na Obr. C.1.



Obr. C.1: Dialogové okno programu CDT E2.

Pokud program narazí na chybu, zobrazí chybovou hlášku. Chybou může být například, že neexistuje vstupní soubor, hrany jsou nepřesně definovány a podobně. Pokud vstupní data obsahují relativně dlouhé hrany, může se při běhu objevit numerická chyba, kterou program neumí řešit. V takovém případě se vypíše příslušná chybová hláška.

V opačném případě program zobrazí hlášku, že vše proběhlo v pořádku a výsledek uloží do výstupního souboru.

Metody v E^3

Obě metody se spouští stejným programem. Následující text popisuje vstupní a výstupní soubor.

Vstupní soubor

Metody samy nevytváří Delaunayovu triangulaci nad vstupními body. Triangulace je načtena ze souboru. Vstupní soubor neobsahuje povinné trojúhelníky, protože metody se neukázaly být použitelné ani pro jeden trojúhelník. Povinný trojúhelník se zadává jako parametr programu. Ukázka vstupního souboru je zde:

```
5
 0.0 -2.1  0.0
-1.0  0.1  1.3
 1.0  0.0  1.0
 0.2  0.1 -1.0
 0.0  2.5  0.1
2
3 2 1 0
1 2 3 4
2
-1 -1 -1 1
-1 -1 -1 0
```

V souboru jsou uloženy po složkách nejprve vstupní body podobně jako je tomu v programu CDT E^2. Poté jsou uloženy tetrahedrony. Každý tetrahedron je popsán indexy 4 bodů. Indexy začínají od nuly. Následují ke každému tetrahedronu jeho 4 sousedi. Sousedi jsou uloženy ve stejném pořadí, jako jsou nadefinovány vrcholy. Ke každému vrcholu se váže index protějšího tetrahedronu k tomuto bodu. Pokud soused neexistuje, je uložena hodnota -1. To znamená, že protější stěna bodu je součástí konvexní obálky. V ukázkovém souboru je například tetrahedron s indexem 0 (první zadefinovaný) tvořen 4 body s indexy 3, 2, 1 a 0. Sousedi sdílející protější trojúhelník k vrcholům 1, 2 a 3 neexistují. Poslední vrchol (bod s indexem 0) se váže k sousedovi 1, což je druhý nadefinovaný tetrahedron.

Výstupní soubor

Výstupní soubor je stejný jako vstupní soubor.

Program se spouští:

```
CDTE3.exe <vstup> <výstup> <t0> <t1> <t2> <metoda>
```

Význam jednotlivých argumentů je následující:

`<vstup>`	jméno vstupního souboru i s cestou
`<výstup>`	jméno výstupního souboru i s cestou
`<t1>`	index prvního bodu tvořící vnucený trojúhelník
`<t2>`	index druhého bodu tvořící vnucený trojúhelník
`<t3>`	index třetího bodu tvořící vnucený trojúhelník
`<metoda>`	číslo metody: 0 … nalezne pouze zasaženou oblast 1 … nalezne zasaženou oblast a provede metodu prohazování trojúhelníků 2 … nalezne zasaženou oblast a provede metodu rekonstrukce oblasti

Výsledkem programu je vždy pouze zasažená oblast. To proto, aby mohla být při experimentech snadno prohlížena. Při použití metod 1 nebo 2 je zasažená oblast přeměněna příslušným algoritmem. Výpisy na konzoli informují o tom, kolik tetrahedronů zasažená oblast obsahuje a zda metoda byla úspěšná nebo selhala. Pokud metoda selhala, je přesto vytvořen výstupní soubor. Ten v takovém případě obsahuje stav, do kterého se metoda dostala. Pokud se v průběhu algoritmu vyskytne singulární případ, který není programem řešen, program vypíše příslušnou hlášku a skončí. V takovém případě výstup není vytvořen.

Prohlížení dat

Prohlížení dat je možné pomocí dvou přiložených programů. Pro 2D data je možné použít program Show_CDT.exe[28]. Pro 3D data je možné použít viewer.exe[29]. 2D prohlížeč se ovládá pomocí dialogového okna. Prohlížeč trojrozměrných dat se spouští následovně:

viewer.exe <jméno souboru s příponou .dat>

Je důležité, aby soubor měl příponu dat. To proto, že prohlížeč umí zobrazovat více druhů vstupů a rozlišuje je pomocí přípony.

[28] Program byl napsán Doc. Dr. Ing. Ivanou Kolingerovou

[29] Program byl napsán Ing. Michalem Varnuškou PhD.

Přehled zkratek

DT Delaunayova triangulace

CDT Delaunayova triangulace s omezením (Constrained Delaunay Triangulation)

E^2 Dvojrozměrný prostor nebo dvojrozměrný případ

E^3 Trojrozměrný prostor nebo trojrozměrný případ