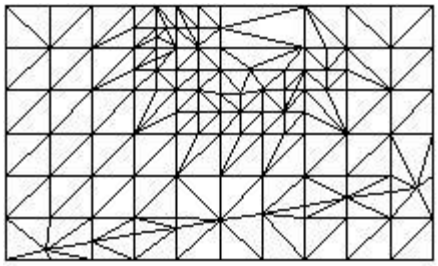
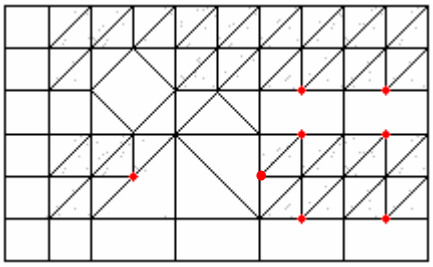
1 Úvod:

Mnoho reálných technických a vědeckých problému se dá popsat parciálními diferenciálními rovnicemi. Již pro 2-dimenzionální (2D) úlohy však lze jen výjimečně nalézt přesné analytické řešení. Proto je nutno původní spojitou úlohu nahradit diskrétní sítí složenou z malých dílků, např. trojúhelníků, tzv. triangulace, a najít přibližné řešení pomocí numerických metod, často vyřešením soustavy algebraických rovnic. Většinou jsou tyto rovnice lineární. Námi požadovanou diskretizační síť můžeme získat pomocí vhodného generátoru.



Obr. 1: Příklad konformní sítě
(definice v kapitole 2.1)



Obr. 2: Příklad nekonformní sítě
s různými prvky a visícími uzly (●)
(definice v kapitole 2.1)

Typicky použijeme vygenerovanou konformní síť, viz Obr. 1 složenou z trojúhelníků či čtyřúhelníků. Vyřešíme úlohu na dané síti vhodnou numerickou metodou. Řešení dosadíme do původní spojité úlohy a odhadneme chybu na dané síti. V případě nevyhovující chyby musíme numerické řešení zpřesnit a toho docílíme tak, že síť zjemníme. Vznikne hustší síť s větším množstvím prvků, a proto i větší množství rovnic. Výpočet se opakuje a je náročnější. Navíc před vyřešením nevíme, zda bude chyba již vyhovující, nebo zda naše síť není složená ze zbytečně malých prvků, a proto nebude výpočet zbytečně zdlouhavý a náročný.

Použití předchozího postupu na úlohy s malými geometrickými detaily by vedlo k potížím. Pro dostatečnou přesnost numerického řešení bychom totiž potřebovali velmi jemnou síť čítající ohromné množství prvků. To by nám samozřejmě zkomplikovalo řešení, jelikož bychom dostali velké množství algebraických rovnic. Mnohdy takové množství, že bychom to ani na běžném počítači vyřešit nemohli. Užitečné by samozřejmě bylo snížit počet prvků dané sítě, urychlit řešení, či jej vůbec umožnit, snížením počtu rovnic a přitom zachovat danou přesnost.

V těchto situacích je proto výhodnější nahradit klasickou konformní síť sítí jinou. Využijeme generátoru hierarchické diskretizační sítě. Na začátku vytvoříme síť, která bude mít zvolený minimální počet prvků námi požadovaného jednoduchého tvaru. Nejlépe trojúhelníky, které jsou pro 2D úlohy vhodné z hlediska jednoduchých a přesných výpočtů. Ideálně to budou

trojúhelníky rovnostranné. Čím mají trojúhelníky, ze kterých je složena disretizační síť, menší rozdíl mezi největším a nejmenším úhlem, tím přesnější numerické řešení dostaneme. Po určení prvotní sítě numericky vyřešíme a dosadíme do původní spojité diferenciální úlohy. Lze pak odhadnout chybu na každém z jednotlivých prvků. Tam, kde je odhadovaná chyba nedostačující, provedeme zjemnění dané sítě rozdělením daného prvku na několik podobných menších prvků. V případě trojúhelníku jej rozdělíme na 4 shodné trojúhelníky pomocí středních příček. Ostatní prvky diskrétní sítě, které mají chybu vyhovující, mohou zůstat. Takto docílíme požadované přesnosti s mnohem menším počtem prvků a tudíž i rovnic.

Hierarchicky vytvořená síť je ovšem nekonformní, viz Obr. 2, což přináší různá úskalí při řešení soustav lineárních rovnic sestavených na dané síti. Tradiční numerické metody totiž nejsou stavěny například na tzv. visící uzly, viz Obr. 2, to jsou uzly, které nejsou vrcholy všech přiléhajících prvků. Pro získání přesného řešení na hierarchické síti proto musíme použít zvláštní metody řešení. V článku **[Sauter]** se popisuje jak zkonstruovat efektivní metodu konečných prvků pro hierarchickou triangulační nekonformní síť.

V této práci se budeme zabývat generací hierarchické triangulační sítě na 2D množině, která bude velmi podobná plátku ementálu. Tato vytvořená síť pak i pomocí metod z článku **[Sauter]** může sloužit k řešení mnoha reálných technických problémů.

Zbytek práce bude mít následující strukturu: V kapitole 2 se zabýváme definicí a samotnou generací hierarchické diskretizační sítě oblasti. V kapitole 3 je nastíněná samotná implementace problému, použité datové struktury v přiloženém programu a popis jak z programátorského hlediska nahlížet na problém triangulace. V poslední kapitole 4 je uvedeno několik příkladů, náhledů a ukázek různých triangulací i samotného programu.

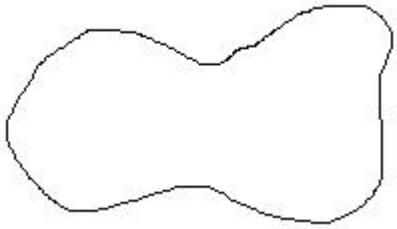
2 Hierarchická triangulační síť

V této kapitole ukážeme, jak může být generována hierarchická triangulační síť.

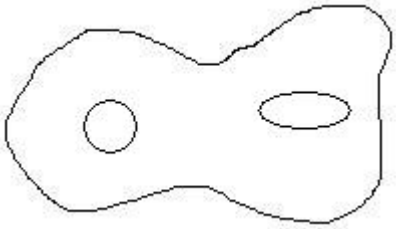
Mějme oblast $\Omega \subset R^2$ s po částech hladkou hranicí $\Gamma := \partial\Omega$. Uvažujeme, že oblast Ω a tím i hranice Γ může obsahovat velké množství mikrostruktur či geometrických detailů. V části 2.1 nadefinujeme použité pojmy, v části 2.2 popíšeme základní postupy generace sítě a v části 2.3 naformulujeme algoritmus generace sítě použitím pseudo-programátorského jazyka.

2.1 Definice pojmů

Oblast Ω: $\Omega \subset R^2$, otevřená množina, vícenásobně souvislá, viz Obr. 4.



Obr. 3: příklad jednoduše souvislé množiny



Obr. 4: příklad vícenásobně souvislé množiny

Hranice oblasti Ω: označme $\Gamma, \partial\Omega$

$$\Gamma := \partial\Omega$$

Uzávěr oblasti Ω: označme $\overline{\Omega}$

$$\overline{\Omega} := \Omega \cup \partial\Omega$$

Element (prvek) sítě K_i: otevřená množinu tvaru trojúhelníku

Diskretizační síť τ_k:

$$\tau_k := \{K_i : i = 1, 2, \ldots, n(\tau_k) \in N\},$$

$$\forall i, j \in \{1, 2, \ldots, n(\tau_k)\} : i \neq j \wedge K_i \cap K_j = \emptyset.$$

Oblast pokrytá sítí τ_k: označme $dom\,\tau_k$, splňuje

$$dom\,\tau_k := \bigcup_{i=1}^{n(\tau_k)} \overline{K_i}\,.$$

Diskretizační síť τ_Ω oblasti Ω: kromě podmínek pro diskretizační síť splňuje navíc:

$$\Omega \subset dom\,\tau_\Omega = \bigcup_{i=1}^{n(\tau_\Omega)} \overline{K}_i .$$

Konformní diskretizační síť τ_Ω oblasti Ω: splňuje navíc:

$\forall K_i, K_j, i \neq j : \overline{K}_i \cap \overline{K}_j = \emptyset$, nebo mají $\overline{K}_i$ a $\overline{K}_j$ právě jeden společný vrchol, nebo mají právě jednu společnou hranu

Zjemnění sítě: zjemněním sítě τ_i rozumíme síť τ_{i+1}, která vznikla rozdělením nejméně jednoho prvku sítě τ_i.

Hierarchická síť $H\tau$ oblasti Ω:

$H\tau := \{ \tau_i : i = 0, \ldots, k_{max} \}$,

kde $k_{\max} \in N$ určuje přesnost diskretizace, síť τ_0 je diskretizační sítí oblasti Ω popsané v části 2.2. Každá síť τ_{i+1} vznikla zjemněním sítě τ_i.

2.2 Popis základních postupů algoritmu

V první fázi generace sítě zvolíme síť τ_0 pro hladinu 0 tak, aby byla splněna podmínka $\Omega \subset dom\,\tau_0$, kde $dom\,\tau_0$ označuje oblast pokrytou sítí τ_0. Pokrytí $dom\,\tau_0$ může být například čtverec či obdélník obsahující celou oblast Ω, který rozdělíme na pěkné trojúhelníky, tj. nejlépe rovnostranné. Můžeme však zvolit i například pravoúhlé trojúhelníky. Dostaneme velmi hrubou síť, která je však velmi špatnou aproximací oblasti Ω.

Předpokládejme, že chceme pomocí hierarchické diskretizační sítě $H\tau$ původní parciální diferenciální rovnice dosáhnout přesnosti odpovídající zjemnění až na hladinu k. Provedeme to postupným zjemňováním sítí τ_k, kde $k \in N$ až do $\tau_{k_{\max}-1}$. Zjemnění sítě τ_k provedeme zjemněním prvků, které mají neprázdný průnik s hranicí Γ. Pokud nějaký takový prvek nalezneme, pomocí středních příček trojúhelník rozdělíme na čtyři shodné trojúhelníky. Každý nově vytvořený trojúhelník bude podobný svému předku a oblast pokrytá čtyřmi novými

trojúhelníky bude odpovídat oblasti pokryté trojúhelníkem, ze kterého vznikly. Dostaneme síť τ_{k+1}, která obsahuje všechny nově vzniklé prvky. Pro pozdější použití, možnosti vyhledávání, vykreslení a dalších užitečných funkcí je nutné poznačit si u každého prvku sítě τ_k zda byl dělen, pokud ne, zda leží vně či uvnitř oblasti Ω. Mohou totiž nastat tyto stavy pro každý z prvků sítě τ_k:

1) *prvek protíná hranici:* Pokud je prvek protínající okraj ze sítě $\tau_{k_{\max}}$, dále jej nedělíme, jelikož vyhovuje naší požadované přesnosti, a proto jej musíme uvést do finální sítě. Pokud je prvek ze sítě τ_k, $k<k_{max}$, musí být dělen a my jej nesmíme zařadit do konečné hierarchické sítě $H\tau$. Zda-li prvek protíná okraj poznáme tak, že oblast prvku obsahuje body, které nenáleží množině Ω, a přitom obsahuje body, které množině Ω náleží:

$K_i \bigcap \Gamma \neq \theta \wedge K_i \bigcap \Omega \neq K_i$.

2) *prvek leží vně oblasti* Ω*:*Prvek nepotřebujeme pro výpočty, dělit jej nepotřebujeme, zobrazit jej můžeme, ale není to potřeba. Mnohdy je zobrazení takových prvků i nežádoucí. Prvek nebude součástí výsledné sítě $H\tau$.

$K_i \bigcap \Omega = \varnothing$

3) *prvek leží uvnitř oblasti* Ω: Tento prvek je pro nás velmi důležitý. Dále jej není třeba dělit a musíme jej uvést do výsledné hierarchické sítě $H\tau$.

$K_i \subset \Omega$

Dále může být velmi užitečné, aby každý prvek obsahoval několik dalších informací. Aby bylo možné dohledat zpětně, z jakého prvku méně přesné sítě byl stvořen, musí každý prvek každé sítě mít svůj vlastní index a krom toho ještě index prvku, jehož zjemněním vznikl. Ve výsledné síti $H\tau$ je také potřeba uvést jakou má prvek hladinu. To může sloužit k sestavení pole všech uzlů, všech hran, či například k vyhledání či zobrazení postupného zjemňování jen určité části oblasti Ω, kterou si zvolíme.

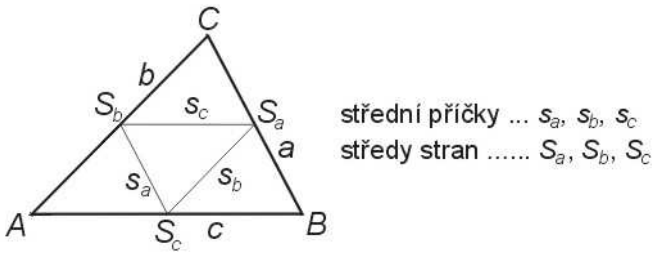
2.3 Algoritmus generování hierarchické sítě τ

V následujícím odstavci se pokusím nastínit algoritmus generace sítě intuitivním popisem podobným programátorským jazykům.

Nechť $H\tau$ je výsledná hierarchická diskretizační síť. Nechť K_i je objekt reprezentující trojúhelník s atributem stav , který popisuje zda protíná hranici, je uvnitř či vně zadané oblasti.

Vstupem je oblast Ω, požadovaná přesnost k_{max}.

procedura generace_mřížky(Ω , k_{max}**);**
začátek
$\tau := \theta$;
k:=0;
τ_0 :=počáteční_diskretizace;
/ Počáteční diskretizace je obdélník obsahující oblast* Ω *rozdělený úhlopříčkou na dva trojúhelníky. */*
dokud ($k < k_{max}$) **dělej**
začátek
$\tau_{k+1} := \theta$;
pro všechna $K_i \in \tau_k$
začátek
zjisti_stav(K_i);
*/*podle průniku s množinou* Ω *a hranicí* Γ *určí stav */*
jestliže(K_i *protíná* Γ)**pak rozděl**(K_i);
/ rozdělí prvek* K_i *pomocí středních příček na 4 nové trojúhelníky, viz Obr. 5, a přidá je do sítě* τ_{k+1} **/*
jestliže$(K_i$ *leží uvnitř oblasti* Ω)**pak přidej**(K_i);
/ přidá prvek* K_i *do sítě* $H\tau$ **/*
konec
k++;
konec
pro všechna $K_i \in \tau_{k_{max}}$
začátek
zjisti_stav(K_i);
jestliže(K_i *neleží vně oblasti* Ω)**přidej**(K_i);
konec
konec



Obr. 5: Trojúhelník ABC rozdělený středními příčkami na trojúhelníky AS_cS_b, S_cS_{aB}, $S_cS_aS_b$ a S_bS_aC. **[Wiki]**

3 Implementace

V této kapitole popíšeme samotnou implementaci problému, datové struktury programu, formát vstupních a výstupních souborů a popíšeme nejdůležitější metody zdrojového kódu. Současná implementace pracuje pouze s obdélníkovou oblastí, ze které vyřízneme kruhové díry. Tyto díry se mohou překrývat.

3.1 Datové struktury

Třída Trojúhelník:

```
class Trojuhelnik
{
public:
  double u1x,u1y,u2x,u2y,u3x,u3y;
  int stav, hladina;
  long index, ipredka;

};
```

Popis atributů:

u1x,u1y: tyto atributy reprezentují souřadnice *x* a *y* uzlu 1. Obdobně pro uzel 2 a uzel 3

stav: tento atribut určuje, v jakém stavu je příslušný trojúhelník vzhledem k oblasti Ω

-1: odpovídá stavu 2) z kapitoly **2.2**, tj. leží vně oblast Ω, např. uvnitř některé ze zadaných děr, kružnic. Trojúhelník nebude dále dělen a nebude zobrazen.

0: odpovídá stavu 1) z kapitoly **2.2**, tj. protíná hranici Γ. Jestliže je trojúhelník z hladiny k_{max}, pak nebude dělen a bude zobrazen, protože již splňuje naši zadanou přesnost. Jestli je z méně přesné hladiny než k_{max}, pak bude rozdělen a nebude zobrazen .

1: odpovídá stavu 3) z kapitoly **2.2**, trojúhelník leží uvnitř oblasti Ω. Trojúhelník nebude dělen a bude zobrazen.

hladina:index hladiny, ve které se trojúhelník nachází.

index: jednoznačně určuje trojúhelník. Žádné dva nemohou mít stejný index.

ipredka: index předka, jehož dělením byl trojúhelník vytvořen.

Třída Kružnice:

```
class Kruznice
{
public:
  double x, y, r;
};
```

Popis atributů:

x,y: tyto atributy reprezentují souřadnice *x* a *y* středu kružnice

r: tento atribut určuje poloměr kružnice

Výsledná hierarchická síť $H\tau$:

```
public: static Trojuhelnik * Tr = new Trojuhelnik[MAX];
```

Hierarchická síť je v programu zastoupena polem objektů třídy Trojuhelnik. Obsahuje všechny trojúhelníky od prvotního rozdělení τ_0 až po nejjemnější síť $\tau_{k_{\max}}$, včetně těch trojúhelníků, které nezahrneme do diskretizační sítě τ zadané oblasti.

V grafické verzi programu tvoříme výslednou diskretizační síť τ uložením do souboru pomocí tlačítka „Ulož jen výslednou síť“.

Pomocí tlačítka „Ulož hierarchickou síť i kružnice“ uložíme do souboru jak všechny použité kružnice, tak i výslednou hierarchickou síť $H\tau$. Všechny atributy trojúhelníků, jednotlivé stavy, hladiny, indexy a indexy předků. Tento soubor může sloužit k analýze chování algoritmu či například k vypsání diskretizační sítě s libovolnou menší hladinou (přesností), než s jakou jsme výsledek ukládali. Také může sloužit pro počítání v budoucnu na stejné oblasti Ω s větším požadavkem přesnosti a to může ušetřit mnoho času při výpočtech.

3.2 Formát vstupního a výstupního souboru

Vstupní soubor: Stěžejní částí programu je samotný algoritmus generace hierarchické sítě. Pro grafickou verzi programu, která má pouze dokázat funkčnost algoritmu, případně názorně ukázat chování sítě při diskretizaci byl zvolen velmi jednoduchý a striktní formát vstupního souboru.

Velikost i tvar oblasti Ω je zadán programem. Pro grafické znázornění je automaticky použit čtverec o délce strany 512 pixelů, který úhlopříčkou rozdělíme na dva rovnoramenné

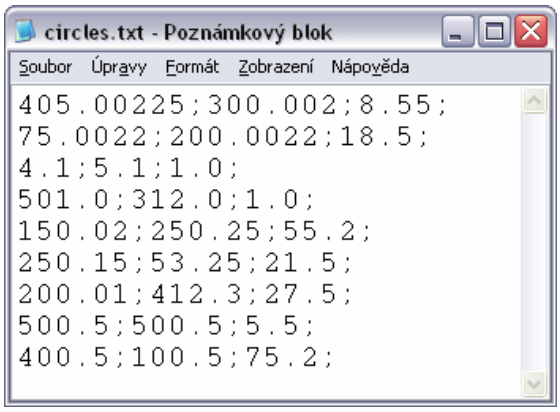
pravoúhlé trojúhelníky, které nám tvoří prvotní rozdělení τ_0. Tento čtverec bude obsahovat díry - kružnice načtené externě ze souboru. Pro použití při vědeckých výpočtech bude samozřejmě umožněno načíst oblast, kterou budeme chtít dále diskretizovat, se všemi potřebnými parametry.

V programu není ošetřeno chybné zadání hodnot, zadané hodnoty musí přesně odpovídat šabloně. Tato šablona je následující:

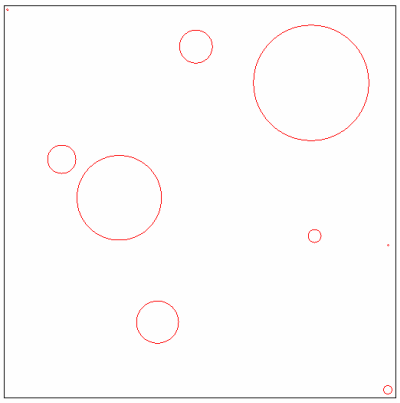
- na jednom řádku je vždy zadána právě jedna kružnice
- hodnoty jsou zadávány v pořadí:
  1. souřadnice *x* středu kružnice
  2. souřadnice *y* středu kružnice
  3. poloměr kružnice
- každá zadaná hodnota musí mít následující tvar: (+ symbolizuje zřetězení)
  ‘celá část’ + ‘. nebo ,’ + ‘desetinná část’ + ‘;’

Například řádek souboru „100,50;151,0;15.0;“ nebo „100.5;151.00;15,0;“ (bez uvozovek) reprezentuje kružnici se středem o souřadnicích [100.5,151] a poloměrem 15.

Náhled vstupního souboru:



Obr. 6: náhled souboru, který určuje oblast použitou v jednom z ukázkových příkladů z kapitoly 4. Obsahuje 9 děr-kružnic.



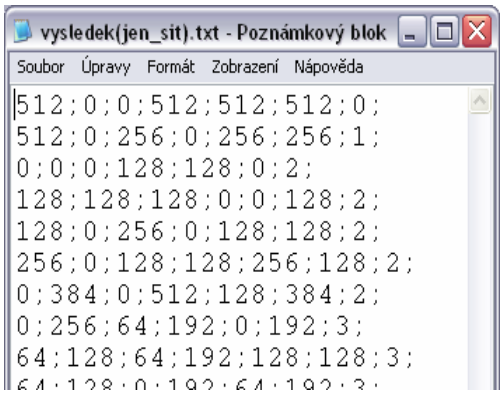
Obr. 7: náhled příslušné oblasti

Výstupní soubor: Máme dvě možnosti jak bude vypadat výstupní soubor.

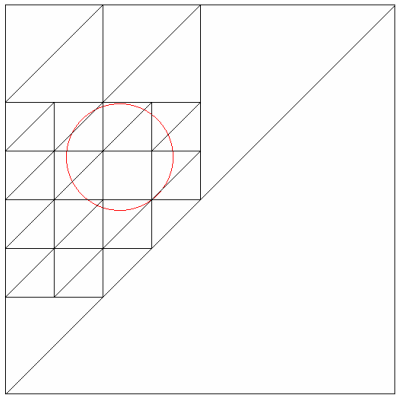
Za prvé: ukládáme pouze výslednou diskretizační síť. V tom případě bude soubor mít tyto vlastnosti:

- na jednom řádku je vždy jeden trojúhelník
- hodnoty jsou udány v pořadí:
  1. souřadnice *x* uzlu 1 příslušného trojúhelníku
  2. souřadnice *y* uzlu 1
  3. souřadnice *x* uzlu 2
  4. souřadnice *y* uzlu 2
  5. souřadnice *x* uzlu 3
  6. souřadnice *y* uzlu 3
  7. hladina, ze které trojúhelník pochází
- každá hodnota má následující tvar: (+ symbolizuje zřetězení)

  ‘celá část’ + ‘. nebo ,’ + ‘desetinná část’ + ‘;’



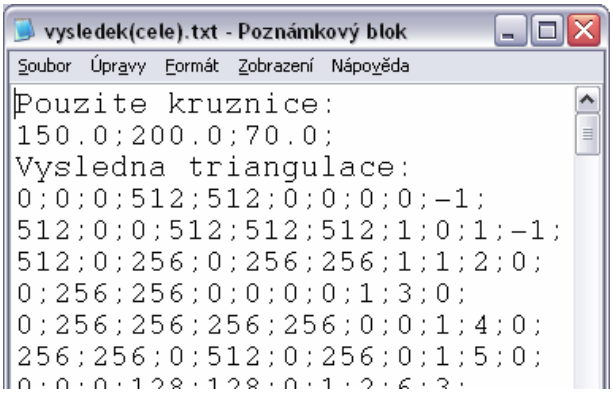
Obr. 8: náhled výsledného souboru, kam jsme uložili pouze výslednou síť potřebnou pro výpočty.
Každý řádek odpovídá jednomu viditelnému trojúhelníku.



Obr. 9: náhled příslušné sítě a oblasti

Za druhé: ukládáme celou hierarchickou triangulační síť $H\tau$ s veškerými potřebnými informacemi. V tom případě bude soubor mít tyto vlastnosti:

- v první část je výpis použitých kružnic (šablona stejná jako u vstupního souboru)
- v druhé části je proveden výpis všech trojúhelníků a jejich atributů
- hodnoty jsou udány v pořadí:
  1. až 6. stejné jako v prvním případě výstupního souboru
  7. stav příslušného trojúhelníku
  8. hladina, ze které trojúhelník pochází
  9. index trojúhelníku
  10. index trojúhelníku, ze kterého vznikl
- každá hodnota je od následující oddělena středníkem



Obr. 10: náhled výsledného souboru, kam jsme uložili celou hierarchickou síť i použité kružnice.

3.3 Popis nejdůležitějších metod

Metoda *triangulace*:

Pro ukázku zde použiji zjednodušenou verzi, která počítá vždy od 0-té hladiny až po zadanou hladinu `hl`. Pokud již máme triangulaci do hloubky menší než `hl`, metoda použitá v programu na dané výsledky naváže a nepočítá od hladiny 0, což znamená značné ušetření času.

```
void triangulace(int hl){
        for(long i=0;i<pocetTrojuhelniku;i++){
              Tr[i].stav=delit(Tr[i]);
                    if(Tr[i].stav==0){
                          if(Tr[i].hladina<hl)rozdelit(Tr[i]);
              };
        }
}
```

Je založena na jednoduchém principu. Metoda *delit*, která je stavebním kamenem algoritmu, určí stav daného trojúhelníku. Je-li třeba trojúhelník rozdělit, zavolá metodu *rozdelit*, která vytvoří 4 nové trojúhelníky.

Metoda *delit*:

Metoda *delit* zjišťuje stav trojúhelníku, čili v podstatě jeho průnik s oblastí Ω. Využívá k tomu metodu *prus*, která hledá průsečíky s jednotlivými kružnicemi, a metodu *obsahujeK*, která zjišťuje, jestli nějaká kružnice neleží uvnitř trojúhelníku. Metodu *obsahujeK* voláme pouze tehdy, pokud není nalezen žádný průsečík. Metoda *delit* v podstatě určuje, kdy máme síť zjemňovat. Při obecnějším použití, např. vykrojené díry z oblasti Ω budou jiného tvaru než kruhy, či důvod zjemnění trojúhelníku bude jiný, stačí zaměnit metodu *delit*. Metoda *triangulace* bude fungovat beze změn.

Metodu *delit* popíši pouze slovně. Postupně porovnává trojúhelník s každou kružnicí. S právě vybranou kružnicí porovnává každou jednotlivou stranu trojúhelníku využitím metody *prus*. Ta zjistí, zda má úsečka s danou kružnici průsečík. Pokud nemá, zjistí také, je-li úsečka uvnitř kružnice či vně. Tato informace je důležitá při určení stavu celého trojúhelníku.

Pokud metoda *delit* zjistí, že trojúhelník leží uvnitř některé z kružnic, tzn. všechny tři strany leží uvnitř té samé kružnice, neporovnává dále trojúhelník s dalšími kružnicemi a okamžitě vrací hodnotu **-1**. To znamená, že je trojúhelník vně oblasti. Pokud neleží uvnitř žádné

z kružnic, zjistíme zda-li byl nalezen průsečík. Pokud ano, vracíme hodnotu **0**. Pokud nenastane ani jeden z případů, musí trojúhelník ležet uvnitř oblasti Ω. Metoda vrací hodnotu **1**

Metoda *delit* počítá i s případem, že se nám některé dvě či více kružnic protíná či překrývá. Pokud bychom měli jistotu, že takový případ nenastane, můžeme metodu urychlit tím, že budeme vracet hodnotu **0** ihned jak zjistíme, že byl nalezen průsečík.

Metoda *prus*:

Používá základních vzorců analytické geometrie **[Algebra]**, jako např. vzdálenost dvou bodů. V první fázi spočteme vzdálenost obou vrcholů úsečky od středu kružnice, porovnáme s poloměrem kružnice, a pokud je jedna vzdálenost větší než poloměr a druhá menší, pak existuje průsečík kružnice a úsečky. Pokud tento případ nenastane, opět porovnáme obě vzdálenosti s poloměrem kružnice, pokud jsou obě menší, úsečka leží uvnitř kružnice. Pokud však nenastane ani jeden z těchto dvou snadno zjistitelných případů, čili oba dva body úsečky leží mimo kružnici, neznamená to, že neexistuje průsečík, úsečka totiž může být sečnou kružnice. V takovém případě metoda spočítá vzdálenost středu kružnice od přímky zadané vrcholy strany trojúhelníku, body $U_1 = [x_1,y_1]$ a $U_2 = [x_2,y_2]$, vztahem:

$$vzd = \frac{|ax_S + by_S + c|}{\sqrt{a^2 + b^2}},$$

kde bod $S = [x_S,y_S]$ je střed kružnice a $ax + by + c = 0$ je obecná rovnice přímky p, kde a a b jsou složky normálového vektoru n přímky p. Složky normálového vektoru zjistíme takto: $a = y_1 - y_2$, $b = x_2 - x_1$, $c=(a*x_1 + b*y_1)* (-1)$. Pokud je vzdálenost vzd větší než poloměr, je jisté, že přímka p úsečka kružnici neprotíná.

Zbývá už jen možnost, že přímka p je opravdu sečnou kružnice. Jen je třeba zjistit, zda protíná kružnici i strana trojúhelníku. Sestavíme obecnou rovnici $rx + sy + t = 0$ přímky q která je kolmá k přímce p a prochází středem kružnice. Spočteme složky r a s normálového vektoru m přímky q: $r = b*(-1)$, $s = a$, $t=(r*x_S + s*y_S)* (-1)$. Vyřešením soustavy dvou obecných rovnic přímek p a q nalezneme souřadnice x a y průsečíku těchto dvou přímek. V této chvíli určíme, leží-li průsečík na úsečce tak, že porovnáme jeho vzdálenosti od obou vrcholů s délkou úsečky. Pokud jsou obě vzdálenosti menší než délka úsečky, pak bod leží na úsečce, a proto kružnice protíná úsečku.

Metoda *obsahujeK*:

Tato metoda má za úkol zjistit, neleží-li některá z kružnic uvnitř trojúhelníku. Vzhledem k tomu, že metodu voláme až poté, co jsme zjistili, že trojúhelník nemá žádný průsečík s kružnicemi, stačí zjistit, jestli neleží uvnitř trojúhelníku střed některé z kružnic. Díky internetové stránce **[Builder]** používáme vcelku jednoduché řešení. Pro každý vrchol sestavíme dvojici vektorů, a vektorovým součinem spočteme souřadnici *z* normály plochy určené těmito vektory. Pokud budou mít souřadnice *z* všech tří normál pro každou dvojici vrcholů stejné znaménko, bude střed ležet uvnitř trojúhelníku. Dvojici vektorů sestavíme následujícím postupem: vektor *a* bude směřovat z uzlu 1 $U_1 = [x_1,y_1]$ do uzlu 2 a $U_2 = [x_2,y_2]$. Vektor *b* pak z uzlu 1 do středu kružnice $S [x_S,y_S]$.

$a_x = x_2 - x_1$; $a_y = y_2 - y_1$;

$b_x = x_S - x_1$; $b_y = y_S - y_1$;

Spočítáme $z_1 = a_x * b_y - a_y * b_x$.

Podobně spočítáme z_2 pro uzly 2 a 3 ($U_2 \rightarrow U_3, U_2 \rightarrow S$) a z_3 pro uzly 3 a 1 ($U_3 \rightarrow U_1, U_3 \rightarrow S$).

Bude-li ležet střed některé kružnice uvnitř trojúhelníku, nastavíme jeho stav jakoby protínal některou kružnici přímo.

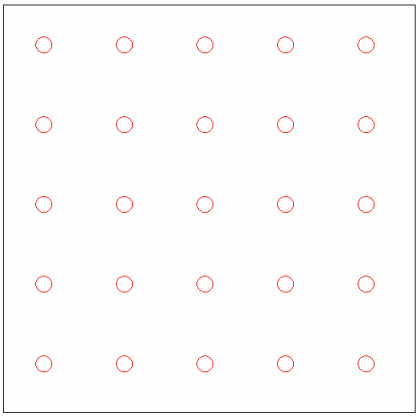
```
bool obsahujeK(Trojuhelnik troj){
	double z1,z2,z3;
	double x1,x2,x3,y1,y2,y3;
	x1=troj.u1x; x2=troj.u2x; x3=troj.u3x;
	y1=troj.u1y; y2=troj.u2y; y3=troj.u3y;
	for(int i=0;i<pocetKruznic;i++){
		z1=(x2-x1)*(Kr[i].y-y1)-(y2-y1)*(Kr[i].x-x1);
		z2=(x1-x3)*(Kr[i].y-y3)-(y1-y3)*(Kr[i].x-x3);
		z3=(x3-x2)*(Kr[i].y-y2)-(y3-y2)*(Kr[i].x-x2);
		if((z1<=0)&&(z2<=0)&&(z3<=0))return(true);
		if((z1>=0)&&(z2>=0)&&(z3>=0))return(true);
	}
	return(false);
}
```

4 Příklady

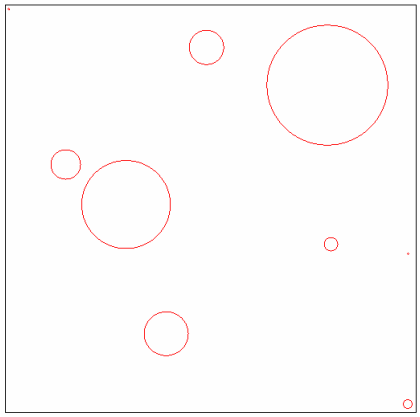
V této kapitole ukážeme několik příkladů diskretizace, vzhled programu a popíšeme jeho funkce.

4.1 Postupné zjemňování

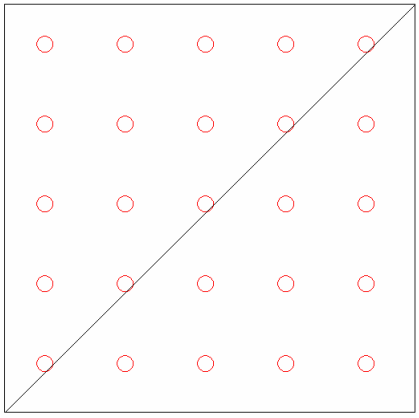
Zde je ukázka několika kroků postupného zjemňování sítě na dvou zvolených oblastech. Pro názornou ukázku jsou vybrány dvě oblasti. Jedna obsahuje 25 stejně velkých symetricky rozmístěných děr a druhá obsahuje 9 různých, od téměř neviditelných po zcela zřetelné. Na Obr. 11 až Obr. 18 jsou zobrazeny náhledy obou oblastí, jejich počáteční dělení a triangulace hloubky 4 a 10.



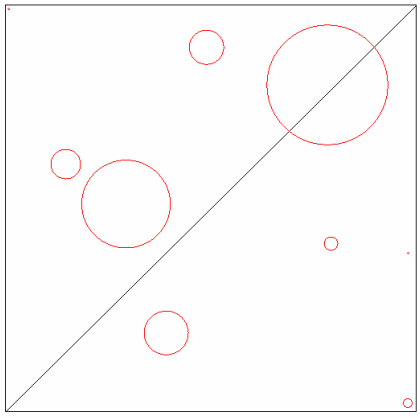
Obr. 11: příklad oblasti o 25-ti dírách



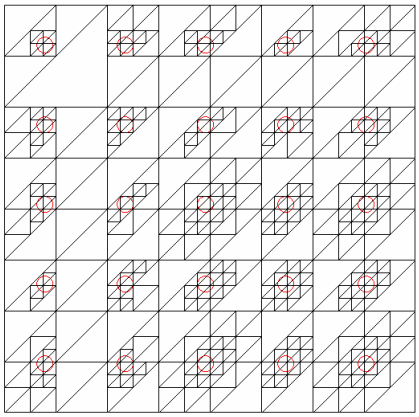
Obr. 12: oblast mající 9 děr



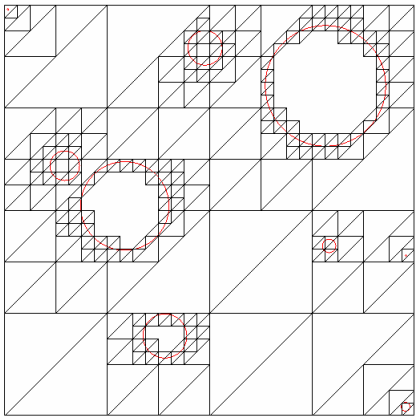
Obr. 13: počáteční rozdělení, hladina 0



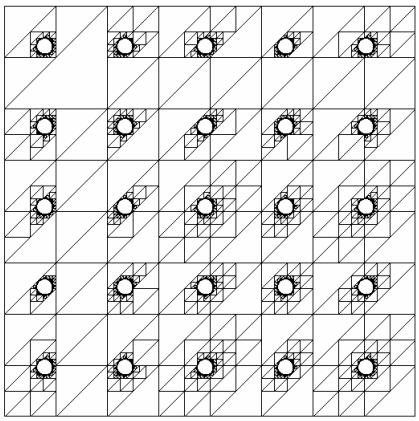
Obr. 14: počáteční rozdělení, hladina 0



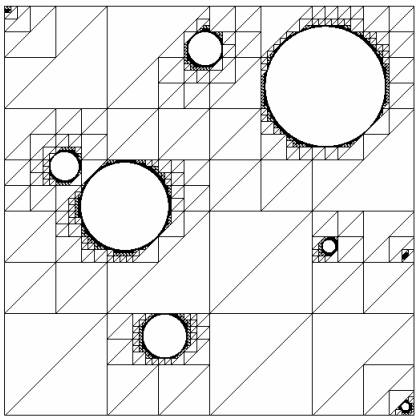
Obr. 15: triangulace – hladina 4



Obr. 16: triangulace – hladina 4

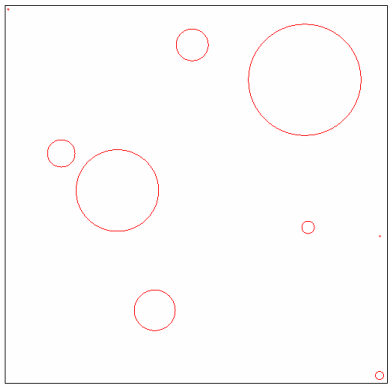


Obr. 17: triangulace – hladina 10

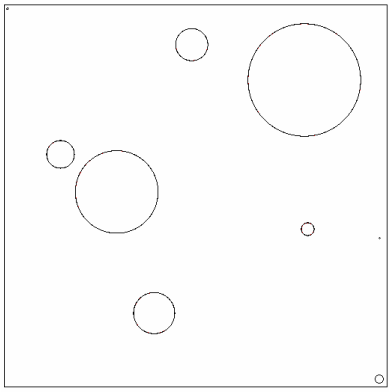


Obr. 18: triangulace – hladina 10

Při triangulaci hladiny 10, Obr. 17 a Obr. 18, kde už nejsme schopni normálním okem zahlédnout nejmenší trojúhelníky, nejjemnější vrstvu. Však pro vědecké účely a dostatečně přesné výpočty můžeme potřebovat triangulovat síť ještě mnohem přesněji.



Obr. 19: náhled oblasti

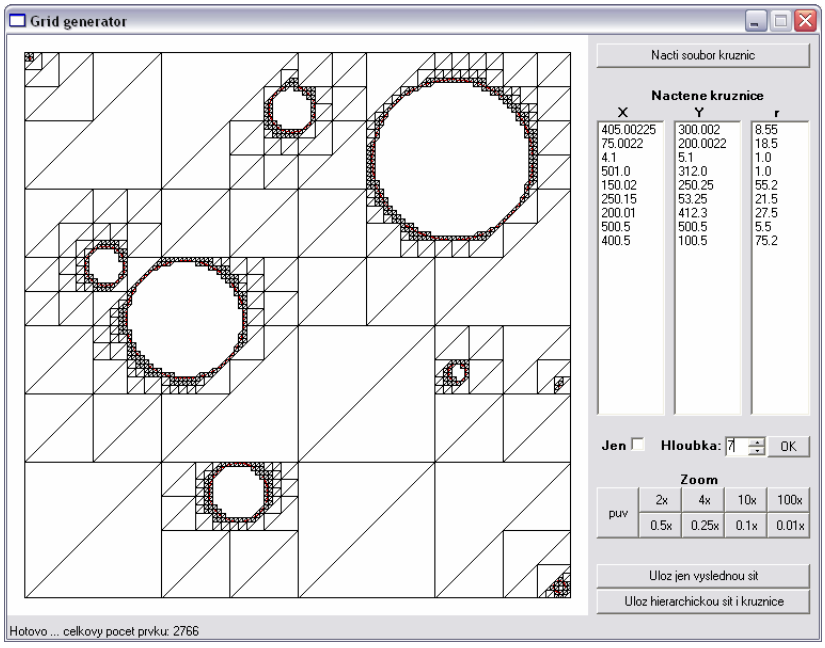


Obr. 20: zobrazení hladiny 14

Na obrázku 19 je zobrazena původní oblast, na obrázku 20 je pak zobrazena pouze nejjemnější hladina, v tomto případě hladina 14. Pouhým okem bychom těžko hledali mezi oběma obrázky rozdíly. Však na levém obrázku máme zobrazeno 9 kružnic, a na pravém přesně 187032 trojúhelníků. Pokud by námi zvolená oblast o délce strany 512(pixelů) představovala čtverec o délce 512 m, měly by trojúhelníky z hladiny 14 ramena dlouhé 3,125 cm a přeponu dlouhou 4,42 cm. Na těchto obrázcích, kdy jedna strana má přibližně 5,12 cm je proto pochopitelné, že nevidíme žádný rozdíl. Strany zobrazených trojúhelníků mají 3,125 a 4,42 μm.

4.2 Grafické uživatelské rozhraní

V této části kapitoly je náhled grafického generátoru hierarchické triangulační sítě, viz Obr. 21, a popis jednotlivých tlačítek.



Obr. 21: náhled spuštěného programu

Tlačítko „Nacti soubor kruznic“

Po kliknutí na tlačítko se nám otevře okno, kde vybereme příslušný textový soubor se zadanými kružnicemi. Ihned se nám zobrazí načtené kružnice a vytvoří se dva trojúhelníky jako prvotní rozdělení oblasti τ_0 .

Nastavení hladiny „hloubka“

Při načtení souboru kružnic se automaticky hloubka nastaví na 0. Nastavení hodnoty se provede triangulace až do příslušné hloubky a aktualizuje se vykreslení. K aktivaci stačí nastavit hodnotu *„hloubka“* pomocí kliknutí na šipku nahoru a dolů či přímo napsáním hodnoty do políčka a stisknutím klávesy ENTER či kliknutím na tlačítko *„OK“*.

Checkbox „Jen“

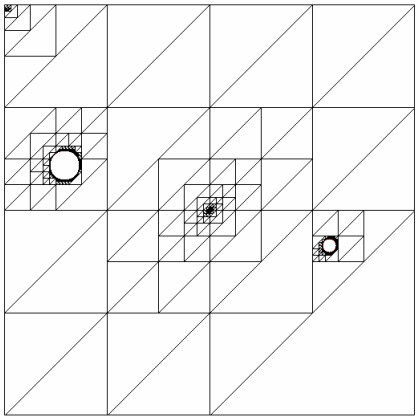
Při zaškrtnutí políčka se bude zobrazovat pouze zvolená hladina. Při nezaškrtnutém políčku se zobrazí celá hierarchická síť odpovídající námi požadované diskretizaci oblasti.

Tlačítka ovládající zoom

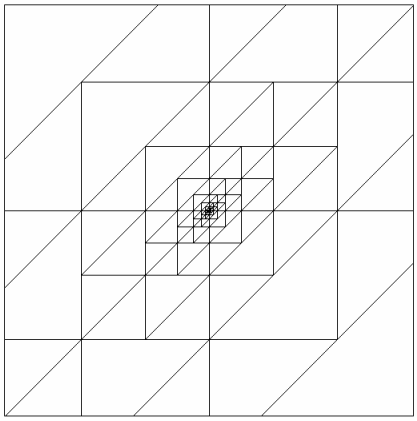
Tlačítko „puv“ navrací zoom na hodnotu 1x a vycentruje vykreslení. Pokud chceme obraz vycentrovat na jiný bod, než je aktuální, stačí na příslušný bod kliknout na vykreslovací plochu. Tlačítky 2x, 4x, 10x, 100x pak zvětšujeme zoom pro dané centrování či tlačítky 0,5x, 0,25x, 0,1x, 0,01x pak zoom zmenšujeme. Příklady na Obr. 22 až Obr.25 pro zoom 1x, 10x, 1000x a 10000x. Kružnice kterou zvětšujeme má poloměr 0,02.

Tlačítka k uložení do souboru:

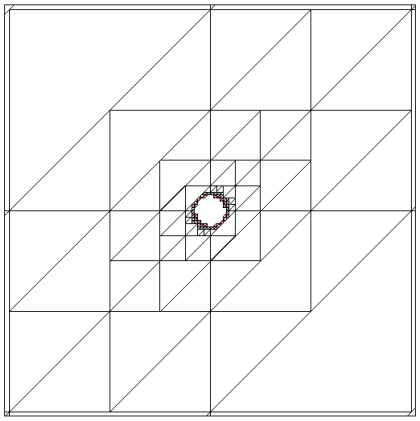
Máme dvě možnosti jak uložit soubor. Podrobnosti jsou uvedeny v kapitole **3.2(**Výstupní soubor).



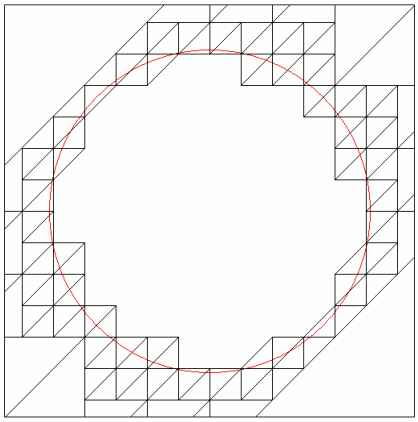
Obr. 22: náhled oblasti, zoom 1x



Obr. 23: zoom 10x, stále kružnici nevidíme



Obr. 24: zoom 1000x, již vidíme kružnici



Obr. 22: zoom 10000x, zřetelně vidíme i ty nejmenší detaily triangulace hloubky 17

5 Závěr

V této práci se mi podařilo sestavit fungující grafický program triangulující obdélníkovou oblast zadanou souborem kružnic definujícím její přesný tvar podobný plátku Ementálu. Samotný stěžejní algoritmus však je postaven tak, že nepotřebuje mít jen pěknou čtvercovou oblast, ale stačí mu jakákoliv počáteční síť a soubor kružnic. Proto se dá po minimální úpravě zdrojového kódu aplikovat na obecnější oblasti. Jiří Lippa, student ze stejné skupiny jako já vyvinul program, který metodou vrstvení trianguluje oblast bez děr, jejíž okraj je zadán úsečkami a Beziérovými křivkami. Spojením obou programů budeme moci vytvořit diskretizační síť určité přesnosti i pro oblast zadanou svým okrajem a souborem děr.

Při testování programu, kdy jsem triangulovaI oblast do velkých hloubek, trvala generace sítě již několik sekund. Proto mě napadlo, že bych mohl daný problém triangulace paralelizovat, čímž bych umožnil tvorbu diskretizační sítě velké přesnosti pro složité oblasti v příznivějším čase. Až se mi podaří problém efektivně paralelizovat, budeme moci generovat síť pro obrovské a velmi složité oblasti, na které by jeden počítač nestačil.

Další motivací pro zlepšení programu je tvorba různých metod pro díry různého tvaru. Prozatím program funguje jen s kružnicemi, ale rád bych jej zobecnil i na díry obecnějšího tvaru. Jak je popsáno v kapitole 3 stačí mi nahradit, případně rozšířit metodu *delit*.

Problémem generace triangulační sítě a řešením reálných problému pomocí metod určeným pro takové sítě bych se rád zabýval i v pozdějším studiu na vysoké škole. Tato problematika mě velmi zaujala a doufám proto, že nezůstane jen u této podoby bakalářské práce, ale vytvořený program se podaří vhodně zoptimalizovat či rozšířit a bude hojně využíván pro potřeby katedry.

Seznam použité literatury:

[Sauter] W. Hackbusch, S.A. Sauter - Composite finite elements for problems containing small geometric details, Computing and Visualization in Science 1, 1997

[Algebra] Zdeněk Dostál - LINEÁRNÍ ALGEBRA, skripta VŠB – TUO, 2004

[Wiki] http://cs.wikipedia.org/

[Builder] http://www.builder.cz/art/cpp/kolize1.html

Seznam příloh:

K Bakalářské práci je přiloženo označené CD se zdrojovými a testovacími soubory programu.