# Manuál k aplikácií a knižnici cheetah

Peter Mitura

*Cheetah* je aplikácia a knižnica umožňujúca efektívne riešiť problém konvexnej obálky. Tento manuál popisuje všetky možnosti aplikácie, funkcie knižnice a postup ich inštalácie.

## Obsah

# 1 Inštalácia

Popis inštalačných procedúr, potrebných k sprevádzkovaniu projektu *Cheetah* na vašom počítači.

## 1.1 Požiadavky

Projekt je určený pre systém GNU/Linux, prípadne iné unix-like systémy. Pre inštaláciu je ďalej potrebné, aby ste na systéme mali nainštalované nasledovné komponenty:

- **GCC** verzia 5.0 alebo vyššia, s podporou OpenMP (`https://gcc.gnu.org/`)
- **GNU ar** (`https://sourceware.org/binutils/docs/binutils/ar.html`)
- **GNU Make** (`https://www.gnu.org/software/make/`)
- *(voliteľne)* **Google Test** (`https://github.com/google/googletest`)

Google Test je potrebný len k sprevádzkovaniu modulu jednotkových testov, aplikácia aj knižnica môže fungovať aj bez neho.

## 1.2 Kompilácia

Kompilácia je automatizovaná nástrojom GNU Make, na výber sú tri direktívy, ktorými sa kompilujú rôzne moduly projektu:

- `make`, `make app` – Aplikácia pre príkazový riadok, vytvorený je spustiteľný súbor `cheetah` v adresári `bin`
- `make lib` – Knižnica, do adresára `bin` je skompilovaný súbor `libcheetah.a` a vytvorený adresár `include`, obsahujúci príslušné hlavičkové súbory.
- `make test` – Jednotkové testy, po kompilácií sú automaticky aj spustené a vyhodnotené.

Upozorňujeme, že každý z týchto modulov používa zdieľané objektové súbory v adresári `build`. Je preto nutné pri každej zmene modulu zavolať príkaz `make clean`, ktorý vráti projekt do počiatočného stavu.

# 2 Aplikácia

Aplikácia sa po kompilácií nachádza v adresári `bin` pod názvom `cheetah`, je možné ju ďalej skopírovať do jedného z adresárov definovaných v premennej prostredia `PATH` a používať z ľubovoľného miesta v systéme.

## 2.1 Formát vstupu

Program prijíma textový vstup, začínajúci prirodzeným číslom $n$, označujúcim počet vstupných bodov, za ktorým nasleduje $n$ dvojíc alebo trojíc reálnych čísel, reprezentujúcich súradnice bodov v $\mathbb{R}^2$ resp. $\mathbb{R}^3$. Čísla sú oddelené bielymi znakmi. Príklad vstupu v $\mathbb{R}^2$:

```
4
0 10
-10 0
9.8002 0
0 1
```

## 2.2 Formát výstupu

Výstup v $\mathbb{R}^2$ má rovnakú podobu, ako vstup. Body v ňom sú vypísané v poradí, v ktorom sa objavujú na obálke, teda proti smeru hodinových ručičiek.

Výstup v $\mathbb{R}^3$ je odlišný, začína sa prirodzeným číslom $n$, označujúcim počet stien. Za ním nasleduje popis $n$ stien, pri čom popis jednej steny sa skladá z prirodzeného čísla $m$ a po ňom $m$ trojíc reálnych čísel, predstavujúcich súradnice vrcholov na danej stene. Vrcholy v rámci jednej steny sú zoradené proti smeru hodinových ručičiek

## 2.3 Základné použitie

Bez špecifikovaných prepínačov aplikácia hľadá konvexnné obálky v $\mathbb{R}^2$ s použitím algoritmu Quickhull. Vstupné dáta sú očakávané na štandardnom vstupe, výstup je vypísaný na štandardný výstup. Ak by sme vyššie uvedený príklad vstupu uložili do súboru `data.in`, použitie by mohlo vyzerať nasledovne:

```
$ bin/cheetah < data.in
3
9.8002 0
-10 0
0 10
```

Funkčnosť je ďalej možné rozšíriť prepínačmi, ktoré popíšeme v nasledujúcich sekciach.

## 2.4 Vstup zo súboru

Prepínač `-i <názov súboru>` umožní načítať vstup z určeného súboru. Príklad použitia:

```
$ bin/cheetah -i data.in
3
9.8002 0
-10 0
0 10
```

## 2.5 Výstup do súboru

Prepínač `-o <názov súboru>` presmeruje výstup zo štandarného výstupu do určeného súboru. Príklad použitia:

```
$ bin/cheetah -i data.in -o data.out
$ cat data.out
3
9.8002 0
-10 0
0 10
```

## 2.6 Meranie času

Prepínač `-t` na štandardný výstup čas, ktorý zaberie nájdenie obálky. Do času sa nezarátava doba potrebná k načítaniu vstupu, alebo vypísaniu výstupu. Príklad použitia:

```
$ bin/cheetah -i data.in -o data.out -t
Execution time: 0.000160927 s.
```

## 2.7 Použitie konkrétneho algoritmu

Štandardným algoritmom pre riešenie je Quickhull. Na zvolenie iného algoritmu pre riešenie je možné použiť prepínač `-s <názov algoritmu>`. Dostupné voľby v $\mathbb{R}^2$ a ich asymptotické zložitosti sú:

- `jarvis` – Jarvis March ($\mathcal{O}(nh)$)
- `graham` – Graham Scan ($\mathcal{O}(n \log n)$)
- `quickhull` – Quickhull ($\mathcal{O}(nh)$ najhorší prípad, $\mathcal{O}(n \log n)$ v priemere)
- `chan` – Chanov algoritmus ($\mathcal{O}(n \log h)$)
- `andrew` – (*experimentálne*) Andrewov algoritmus ($\mathcal{O}(n \log n)$)

Pre $\mathbb{R}^3$ je dostupný len algoritmus Jarvis March, ktorý je použitý aj v základe.

## 2.8 Voľba dimenzie

V základe program používa rovinné algoritmy, dimenziu vstupu je možné špecifikovať prepínačom `-d <dimenzia>`. Súčasne podporované hodnoty sú 2 a 3.

## 2.9 Paralelizácia

Aplikácia štandardne púšťa sekvenčné verzie algoritmov, paralelizáciu je možné zapnúť prepínačom `-m <počet vlákien>`. Funguje len v $\mathbb{R}^2$ a maximálny počet vlákien je z dôvodu ochrany stability systému limitovaný na 24. Príklad použitia:

```
$ bin/cheetah -t -o /dev/null -i bigdata.in -s graham
Execution time: 0.700449 s.
$ bin/cheetah -t -o /dev/null -i bigdata.in -s graham -m 2
Execution time: 0.467592 s.
```

## 2.10 Test výkonu

Výkon je mimo prepínača `-t` možné otestovať aj priloženým generátorom dát, ktorý dokáže vytvoriť vstup s daným počtom bodov vo vstupnej množine a na obálke. Test sa spúšťa prepínačom `-p <n> <h> <s> <r> <t>`, kde $n$ je počet bodov vo vstupnej množine ($n \in \mathbb{N}, n < 2^{31}$), $h$ počet bodov na obálke ($h \in \mathbb{N}, h \leq n$), $s$ určuje rozsah súradníc bodov $[-s, s]$ ($s \in \mathbb{N}$), $r$ počet zopakovaní testu ($r \in \mathbb{N}$) (čas sa ráta ako súčet všetkých behov) a $t$ počet vlákien ($t \in \mathbb{N}, 1 \leq t \leq 24$).

Upozorňujeme, že pri veľkom množstve bodov na obálke (typicky nad $5 \cdot 10^4$) sa začne prejavovať nepresnosť dátového typu `double` a na nájdených obálkach sa začne objavovať nižší počet bodov ako bol zadaný, čo môže ovplyvniť aj časy výpočtu. Testovač na túto skutočnosť upozorní varovaním. Príklad:

```
$ bin/cheetah -p 10000000 10 1000 1 1
Generating test instance...
...done, running test
time: 0.586616 s
$ bin/cheetah -p 10000000 10000 1000 1 1
Generating test instance...
...done, running test
time: 1.82755 s
$ bin/cheetah -p 10000000 1000000 1000 1 1
Generating test instance...
...done, running test
[WARNING] Precision errors occured (h may be too high)
time: 2.60803 s
```

# 3 Knižnica

Modul knižnice umožňuje pridať implementované algoritmy priamo do programov napísaných v jazyku C++. Oproti aplikácií v nej chýbajú výskumne založené vymoženosti ako testovanie rýchlosti, obsahuje však plné možnosti ohľadne výberu algoritmov a paralelizácie.

## 3.1 Použitie

Knižnica je kompilovaná pre statické linkovanie s cieľovou aplikáciou. Pri kompilácií výsledku je nutné použiť minimálne prepínače `-std=c++11` a `-fopenmp`. Ďalej je nutné direktívami `-I` a `-L` špecifikovať cestu k hlavičkovým súborom a knižnici a prepínačom `-lcheetah` povoliť linkovanie. Funkcie knižnice sa nachádzajú v hlavičkovom súbore `cheetah/core.h`

K lepšiemu pochopeniu poslúži ukážková aplikácia využívajúca našu knižnicu. Do koreňového adresára najprv vložíme skompilovaný súbor `libcheetah.a` a adresár include. Potom vytvoríme nasledovný súbor `main.cpp`:

```
#include <iostream>
#include "cheetah/core.h"

int main() {
    ch::Points2D input, output;
    input.add({10, 0});
    input.add({-10, 0});
    input.add({0, 10});
    input.add({1, 1});
    ch::findHull(input, output);
    std::cout << output.getSize() << std::endl;
    return 0;
}
```

*(Fungovanie jednotlivých funkcií a dátových štruktúr vysvetlíme ďalej v tejto sekcií)*

Kompiláciu a spustenie potom môžeme realizovať nasledovnou sekvenciou príkazov:

```
$ g++ -std=c++11 -fopenmp -I include/ -L . main.cpp -lcheetah -o sample
$ ./sample
3
```

## 3.2 Dátové štruktúry

Pre reprezentáciu množín bodov sú použité dátové štruktúry `ch::Points2D` a `ch::Points3D`. Obe disponujú metódou

```
const std::vector<std::vector<double>>& getData() const;
```

ktorá vráti konštantnú referenciu na ich vnútornú dátovú reprezentáciu, z ktorej je možné čítať ich obsah. Pre pridávanie bodov slúži funkcia:

```
bool add(std::vector<double> point);
```

ktorá vráti `false` v prípade, že sa počet súradníc v posielanom vektore nezhoduje s počtom dimenzií danej množiny (a daný bod potom nie je ani pridaný do množiny).

Nakoniec majú ešte obe štruktúry metódu `getSize`, ktorá vráti počet bodov v nich.

Pre reprentáciu konvexného mnohostena, ktorý je výstupom hľadania obálky v troch rozmerov je štruktúra `ch::Polyhedron`. Tá je reprezentovaná ako `std::vector<ch::Points3D>`, teda ako vektor stien, ktoré sú uložené ako zoznam vrcholov. K tejto reprezentácií sa analogicky dá pristúpiť metódou `getFaces`, pre vkladanie bodov slúži metóda `addFace(ch::Points3D)`.

## 3.3 Funkcie

V tejto sekcií uvedieme zoznam funkcií, ktoré sprístupňuje hlavičkový súbor `core.h` a popis ich použitia. Všetky funkcie sú súčasťou menného priestoru *ch* a je ich potrebné volať so zodpovedajúcou predponou.

### 3.3.1 Nájdenie obálky

Nájdenie obálky v $\mathbb{R}^2$ pomocou algoritmu Quickhull.

```
Points2D& findHull(const Points2D& input, Points2D& output);
```

- **parameter *input*** – Referencia na vstupnú sadu bodov.
- **parameter *output*** – Referencia na výstupnú dátovú štruktúru, do ktorej bude vložený zoznam bodov tvoriacich obálku, zoradenú proti smeru hodinových ručičiek.
- **návratová hodnota** – Rovnaká ako `output`.

### 3.3.2 Nájdenie obálky s voľbou algoritmu

Nájdenie obálky v $\mathbb{R}^2$ pomocou zvoleného algoritmu:

```
Points2D& findHull(const Points2D& input, Points2D& output, SolverType type);
```

- **parameter *input*** – Referencia na vstupnú sadu bodov.
- **parameter *output*** – Referencia na výstupnú dátovú štruktúru, do ktorej bude vložený zoznam bodov tvoriacich obálku, zoradenú proti smeru hodinových ručičiek.
- **parameter *input*** – Označenie zvoleného algoritmu. Dostupné voľby:
  - `ch::JARVIS`
  - `ch::GRAHAM`
  - `ch::QUICKHULL`
  - `ch::CHAN`
  - `ch::ANDREW` *(experimentálny)*
- **návratová hodnota** – Rovnaká ako `output`.

### 3.3.3 Paralelné nájdenie obálky

Pre paralelné nájdenie obálky sú dostupné dve funkcie, analogické s predošlou dvojicou:

```
Points2D& findHullParallel(const Points2D& input, Points2D& output, int thr);

Points2D& findHullParallel(const Points2D& input, Points2D& output,
    SolverType type, int thr);
```

Jediný nový parameter je prirodzené číslo `thr`, ktoré určuje počet použitých vlákien (maximálne ale 24).

### 3.3.4 Nájdenie obálky v 3D

Nájdenie obálky v $\mathbb{R}^3$ pomocou algoritmu Jarvis March:

```
Polyhedron& findHull3D(const Points3D input, Polyhedron& output)
```

- **parameter *input*** – Referencia na vstupnú sadu 3D bodov.
- **parameter *output*** – Referencia na výstupnú dátovú štruktúru, reprezentujúcu konvexný mnohosten.
- **návratová hodnota** – Rovnaká ako `output`.

### 3.3.5 Aproximácia obálky (experimentálne)

Do knižnice sme experimentálne zaradili aj rýchlu aproximáciu konvexnej obálky algoritmom *BFP*. Upozorňujeme teda, že aj keď je tento algoritmus najrýchlejší, správnosť výsledku nie je očakávaná a je bude sa riešeniu len do istej miery blížiť.

```
Points2D& approximateHull(const Points2D& input, Points2D& output)
```

- **parameter *input*** – Referencia na vstupnú sadu bodov.
- **parameter *output*** – Referencia na výstupnú dátovú štruktúru, obsahujúcu približnú podobu obálky.
- **návratová hodnota** – Rovnaká ako `output`.