

České vysoké učení technické v Praze  
Fakulta Stavební



155ADKI Algoritmy digitální kartografie a GIS  
Geometrické vyhľadávanie bodu

Bc. Adriana Brezničanová, Bc. Martin Kouba

22. 10. 2020

## Úloha č. 1: Geometrické vyhledávání bodu

*Vstup:* Souvislá polygonová mapa  $n$  polygonů  $\{P_1, \dots, P_n\}$ , analyzovaný bod  $q$ .

*Výstup:*  $P_i, q \in P_i$ .

Nad polygonovou mapou implementujete následující algoritmy pro geometrické vyhledávání:

- Ray Crossing Algorithm (varianta s posunem těžiště polygonu).
- Winding Number Algorithm.

Nalezený polygon obsahující zadaný bod  $q$  graficky zvýrazněte vhodným způsobem (např. vyplněním, šrafováním, blikáním). Grafické rozhraní vytvořte s využitím frameworku QT.

Pro generování nekonvexních polygonů můžete navrhnut vlastní algoritmus či použít existující geografická data (např. mapa evropských států).

Polygony budou načítány z textového souboru ve Vámi zvoleném formátu. Pro datovou reprezentaci jednotlivých polygonů použijte špagetový model.

### Hodnocení:

Krok	Hodnocení
Detekce polohy bodu rozlišující stavy uvnitř, vně na hranici polygonu.	10b
Ošetření singulárního případu u Winding Number Algorithm: bod leží na hraně polygonu.	+2b
Ošetření singulárního případu u obou algoritmů: bod je totožný s vrcholem jednoho či více polygonů.	+2b
Zvýraznění všech polygonů pro oba výše uvedené singulární případy.	+2b
Algoritmus pro automatické generování nekonvexních polygonů.	+5b
<b>Max celkem:</b>	<b>21b</b>

Čas zpracování: 2 týdny.

## Obsah

1. Popis a rozbor problému .....	4
2. Popis algoritmov.....	5
2.1 Winding Number Algorithm .....	5
2.2 Ray Crossing Algorithm.....	6
3. Vstupné dátá .....	7
4. Výstupné dátá.....	7
5. Printscreeny z vytvorennej aplikácie .....	8
6. Dokumentácia .....	10
6.1 Trieda algorithms.h .....	10
6.2 Trieda draw.h .....	10
6.3 Trieda widget.h.....	10
6.4 algorithms.cpp.....	11
6.5 draw.cpp.....	12
6.6 widget.cpp.....	14
7. Záver .....	15
8. Zdroje.....	15
9. Zoznam obrázkov.....	15

## 1. Popis a rozbor problému

Ako vstup máme danú množinu bodov  $p_i$  tvoriacich vrcholy mnohouholníkov  $P_i$  a bod  $q$ , ktorého polohu voči týmto mnohouholníkom chceme zistiť. Riešime tzv. Point Location Problem. Point Location Problem je vlastne geometrické vyhľadávanie čiže určovanie vzájomnej polohy dvoch ľubovoľných entít (v našom prípade bodu a polygónu) v 2D alebo 3D priestore. Toto využívame napríklad pri zisťovaní polohy kurzoru po kliknutí napríklad do polygónov obcí. Chceme zistiť, do ktorej obce som klikla, prípadne som klikla mimo Českú Republiku. Úlohu môžeme riešiť buď pre konvexné mnohouholníky (jednoduchšie) alebo nekonvexné mnohouholníky (zložitejší postup).

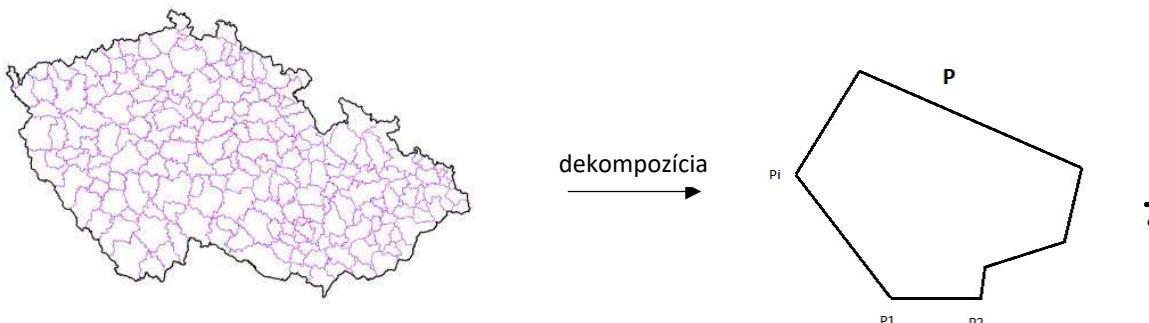
**Na riešenie môžeme využiť 2 stratégie:**

=> beriem polygón po polygóne a určujem polohu bodu  $q$

- výhodná a jednoduchá implementácia (my sme počítali s touto stratégou)
- opakované riešenie lokálnej procedúry => globálna procedúra

*Pozn.* – *lokálna procedúra využíva opakované určenie polohy bodu  $q$  vzhľadom k mnohouholníku a jej výsledok môže byť, že bod  $q$  patrí/ nepatrí/ leží na hrane mnohouholníka; globálna procedúra opakuje lokálnu procedúru pre každý mnohouholník data setu a výsledok môže byť - mnohouholníkov ( $q$  leží mimo všetky mnohouholníky), 1 mnohouholník ( $q$  leží v konkrétnom mnohouholníku) a viac ako 1 mnohouholník (bod  $q$  leží na hrane alebo v uzle susediacich mnohouholníkov).*

=> Trapezoidálne mapy – využíva binárne stromy a je výhodná pre väčšie data sety.

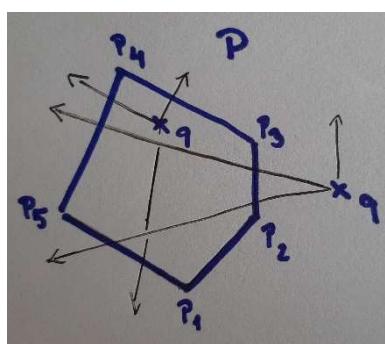


Obrázok 1 - Prevod na lokálnu procedúru

**Lokálne procedúry sa najčastejšie riešia dvoma spôsobmi:**

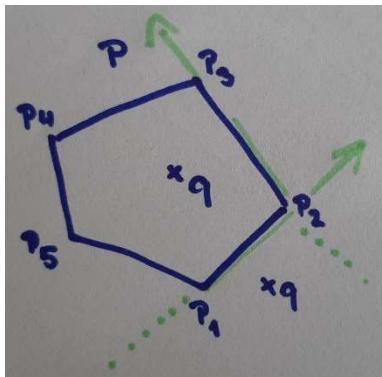
- test polohy bodov voči hrane mnohouholníka (opakovaný Half Plane test)
- lúčový (paprskový) algoritmus (Ray Crossing Algorithm)

**Konvexné mnohouholníky**



Myšlienka je, že viedem polpriamky z bodu  $q$  a pozorujem počet priesecníkov s polygónom  $P$ . Ak sa bod  $q$  nachádza v polygóne  $P$ , počet priesecníkov je nepárny. Ak bod  $q$  leží mimo polygón, počet priesecníkov je párný (zbytok po celočíselnom delení je 0).

Obrázok 2 - Určovanie počtu priesecníkov



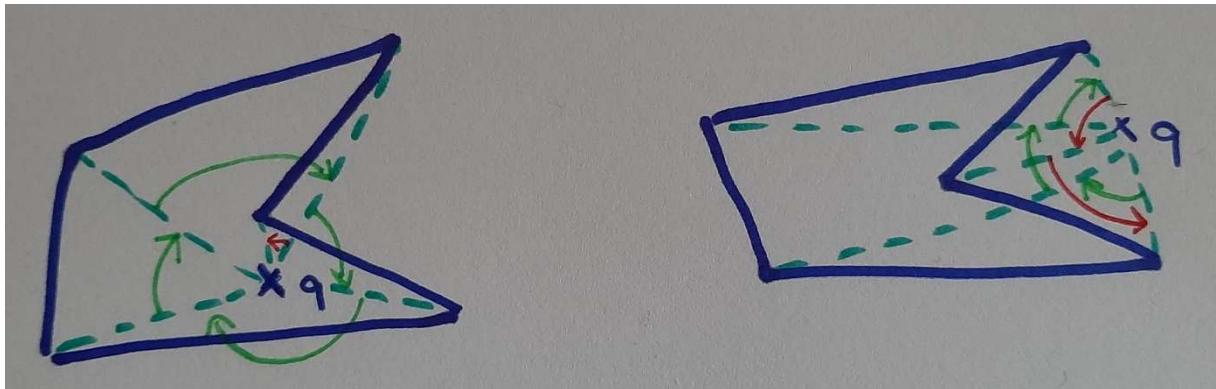
Ďalším riešením je, že vezmem stranu \$P\_1\$ a \$P\_2\$ a viedem ňou polpriamku. My uvažjeme orientáciu proti smeru hodinových ručičiek (CCW - counterclockwise). Určujeme polohu bodu \$q\$ voči polpriamke. Ak je bod \$q\$ v ľavej polrovine, tak leží v mnohouholníku. Ak je bod \$q\$ mimo, polohy voči priamkam sa menia.

Obrázok 3 - Vedenie priamok CCW

## 2. Popis algoritmov

### 2.1 Winding Number Algorithm

Platí pre konvexné aj nekonvexné mnohouholníky. Myšlienka je taká, že pozorovateľ stojí na bode \$q\$ a pozera sa na všetky lomové body polygónu, takže ak \$q \in P\$ musí sa otočiť o uhol \$2\pi\$ (súčet uhlov v absolútnej hodnote). Ak je tento uhol menší než \$2\pi\$, potom \$q \notin P\$.



Obrázok 4 - Určovanie Winding Number

Určíme si testovaciu podmienku:

$$t: \sum_{i=1}^n \omega_i = 2\pi$$

pričom \$\omega\_i\$ vypočítame pomocou dvoch smerových vektorov \$\vec{u} = |P\_1q|\$ a \$\vec{v} = |qP\_2|.

$$\omega = \text{acos} \left( \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \cdot \|\vec{v}\|} \right), \text{ kde } P_1 \text{ a } P_2 \text{ sú vrcholy polygónu a } q \text{ je bod, kt. polohu zistujeme.}$$

Zaujíma nás znamienko determinantu, ktoré rozhoduje v akej polrovine sa bod \$q\$ nachádza (ľavej alebo pravej) alebo je rovný nule a bod sa nachádza na hrane.

$$\det = \begin{vmatrix} u_x & u_y \\ v_x & v_y \end{vmatrix}$$

Winding number \$\Omega\$ sa teda rovná sume uhlov \$\omega\_i\$.

#### Popis algoritmu:

- 1) Inicializácia Winding Number na 0 (\$\Omega = 0\$) a nastavenie tolerancie \$\epsilon\$ blízke nule
- 2) Spustenie *for* cyklu pre všetky trojice \$(p\_i, q, p\_{i+1})\$
- 3) Určenie polohy bodu \$q\$ k príslušnej strane polygónu
- 4) Určenie uhlu \$\omega\$ (uhol, ktorý zviera bod \$q\$ a počiatočným a koncovým bodom príslušnej hrany)

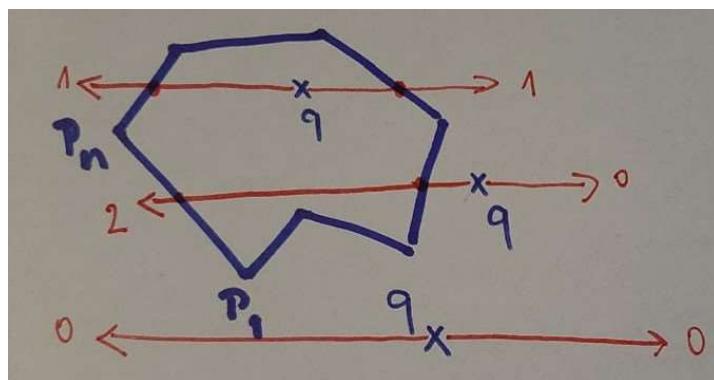
- 5) Príkaz *if* –  $q \in P \left\{ \begin{array}{l} \partial L: \omega > 0 \\ \partial R: \omega < 0 \end{array} \right\}$  (Half Plane Test)
- 6) Príkaz *else* -  $q \notin P$
- 7) Príkaz *if* –  $(|\Omega| - 2\pi) < \epsilon$ , potom  $q \in P$  (test na odchýlku od  $2\pi$ )
- 8) Príkaz *else if* –  $|\Omega| < \epsilon$ , potom  $q \notin P$
- 8) Príkaz *else* – bod leží na hrane polygónu, týmto je odstránená prípadná singularita

## 2.2 Ray Crossing Algorithm

Myšlienka algoritmu je, že vedieme paprsek (polpriamku) z bodu  $q$  nezávisle na smere.

Počítame počet priesčníkov  $k$  s polygónom. Tento paprsek nazývame Ray ( $r$ ).

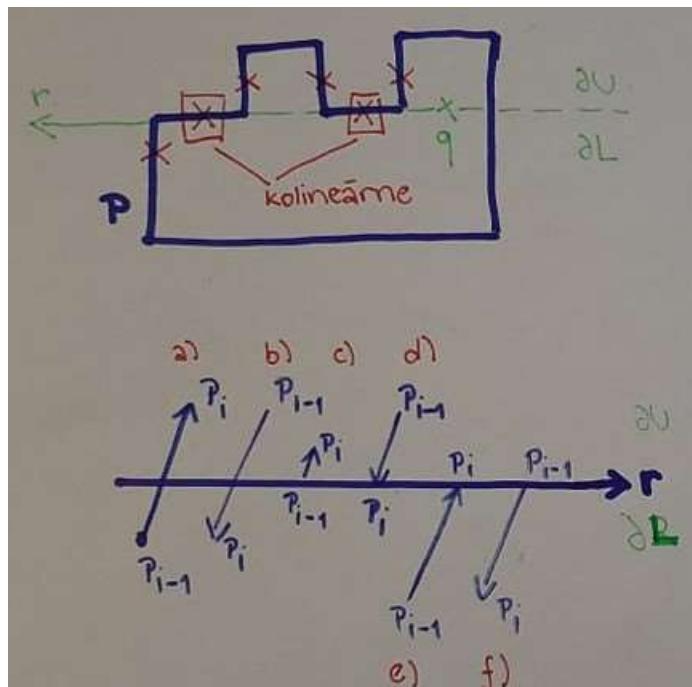
Paprsek vedieme rovnobežne alebo zvisle, tým sa situácia zjednoduší. V našom prípade vedieme rovnobežku s osou  $X$ , ktorá prechádza bodom  $q$  a teda  $r(q)$ :  $y = y_q$ .



Skúmame  $k(r, P)$ %2 -  
 $\begin{cases} 0 : q \notin P, \text{ párne} \\ 1 : q \in P, \text{ nepárne} \end{cases}$

Obrázok 5 - Určovanie počtu priesčníkov (Ray Crossing Alg.)

Tu však tiež narážame na singularity a to, že priesčník je kolineárny (toho sa musíme zbaviť).



Riešenie je rozdelenie polygónu na polroviny – hornú ( $\partial U$  (upper)) a dolnú ( $\partial L$  (lower)).

V prípade vľavo vznikne  $k = 6$  (podľa definície mimo polygón  $P$ ), ale vidíme, že bod je v polygóne.

Obrázok 6 - Rozdelenie polygónu na polroviny

Započítavať budeme teda body splňujúce nasledujúce podmienky:

- z prípadu a) a c)  $\Rightarrow (Y_i > Y_q) \wedge (Y_{i-1} \leq Y_q)$
- z prípadu b) a d)  $\Rightarrow (Y_{i-1} > Y_q) \wedge (Y_i \leq Y_q)$

- testovacou podmienkou (výsledkom) je logický súčet  $(Y_i > Y_q) \wedge (Y_{i-1} \leq Y_q) \vee (Y_{i-1} > Y_q) \wedge (Y_i \leq Y_q)$  pre hornú polrovinu, pre dolnú sa zamenia znamienka.

Zjednodušenie plynie z tvorby vlastného (lokálneho) súradnicového systému – redukcia k bodu  $q$ . Potom redukované súradnice  $x'_i$  a  $y'_i$  jednotlivých bodov sú:

$$x'_i = x_i - x_q$$

$$y'_i = y_i - y_q$$

Počítame priesečník  $M$  osy  $x'$  a hrany, pričom  $y'_{m=0}$  a  $x'_{m=\frac{x'_i y'_{i-1} - x'_{i-1} y'_i}{y'_i - y'_{i-1}}}$ .

Test existencie priesečníku -  $(y'_i > 0) \wedge (y'_{i-1} \leq 0) \vee (y'_{i-1} > 0) \wedge (y'_i \leq 0)$ .

Započítavané sú len priesečníky v pravej polrovine, čiže  $x'_{m=0}$  je kladné.

Tento postup rieši prípadné singularity.

#### **Popis algoritmu:**

- 1) Inicializácia počtu priesečníkov  $k = 0$
- 2) Cyklus *for* cez všetky vrcholy polygónu
- 3) Redukcie súradník  $x'_i = x_i - x_q$  a  $y'_i = y_i - y_q$
- 4) Test existencie priesečníku  $(y'_i > 0) \wedge (y'_{i-1} \leq 0) \vee (y'_{i-1} > 0) \wedge (y'_i \leq 0)$  – vhodný segment
- 5)  $x'_{m=\frac{x'_i y'_{i-1} - x'_{i-1} y'_i}{y'_i - y'_{i-1}}}$  - vhodný priesečník
- 6) Príkaz *if* –  $(k \% 2) \neq 0$ , potom  $q \in P$
- 7) *Else* -  $q \notin P$

### 3. Vstupné dátá

Vstupné dátá sú v textovom súbore, ktorý sa do programu importuje pomocou tlačidla *Import*. Po stlačení tlačidla zadáme cestu k tomuto súboru.

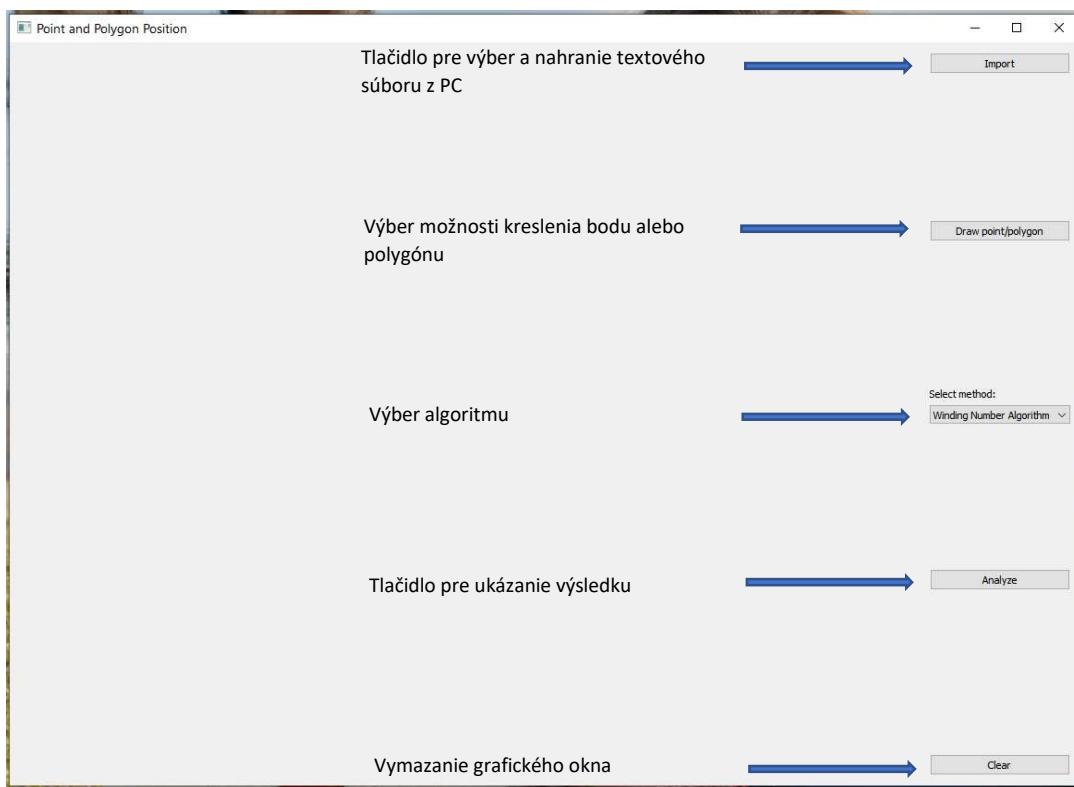
Súbor sa skladá z troch stĺpcov a n riadkov. Riadok začína číslom vrcholu, nasleduje súradnica X a následne Y.

Polygóny je tiež možné ručne kresliť.

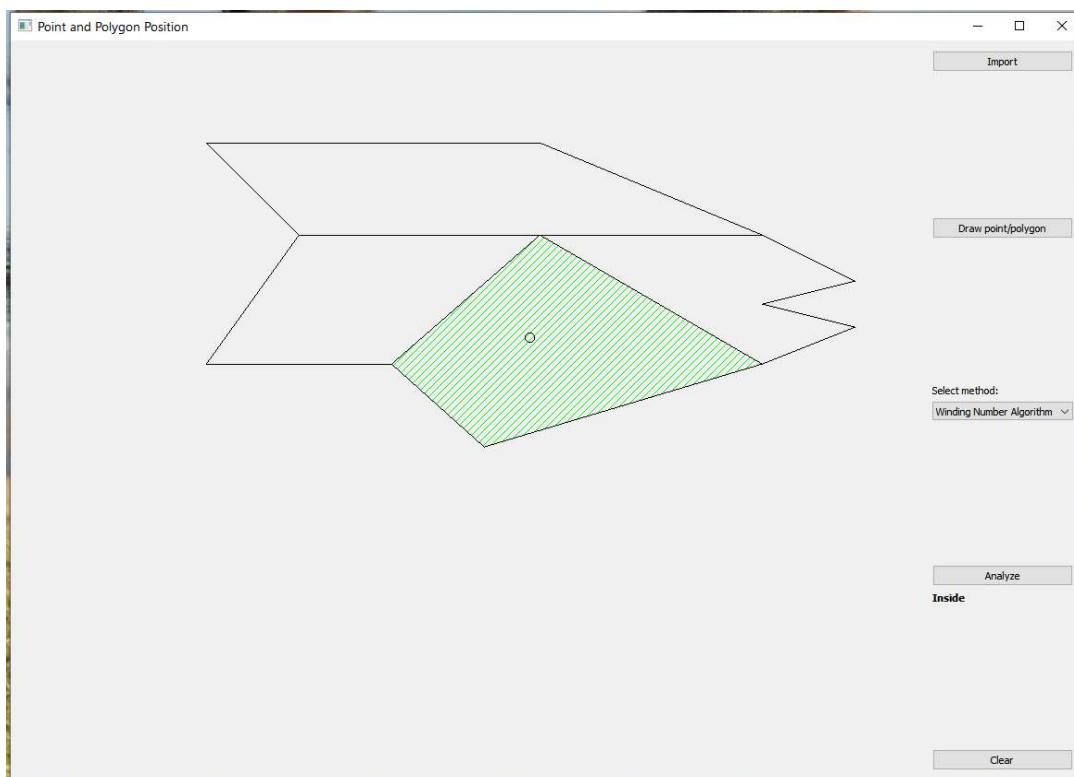
### 4. Výstupné dátá

Výstupom rozumieme vysvetlenie polygónu (v prípade, že bod leží v polygóne) a textový výstup o polohe bodu priamo v grafickom rozhraní aplikácie pod tlačidlom *Analyze*.

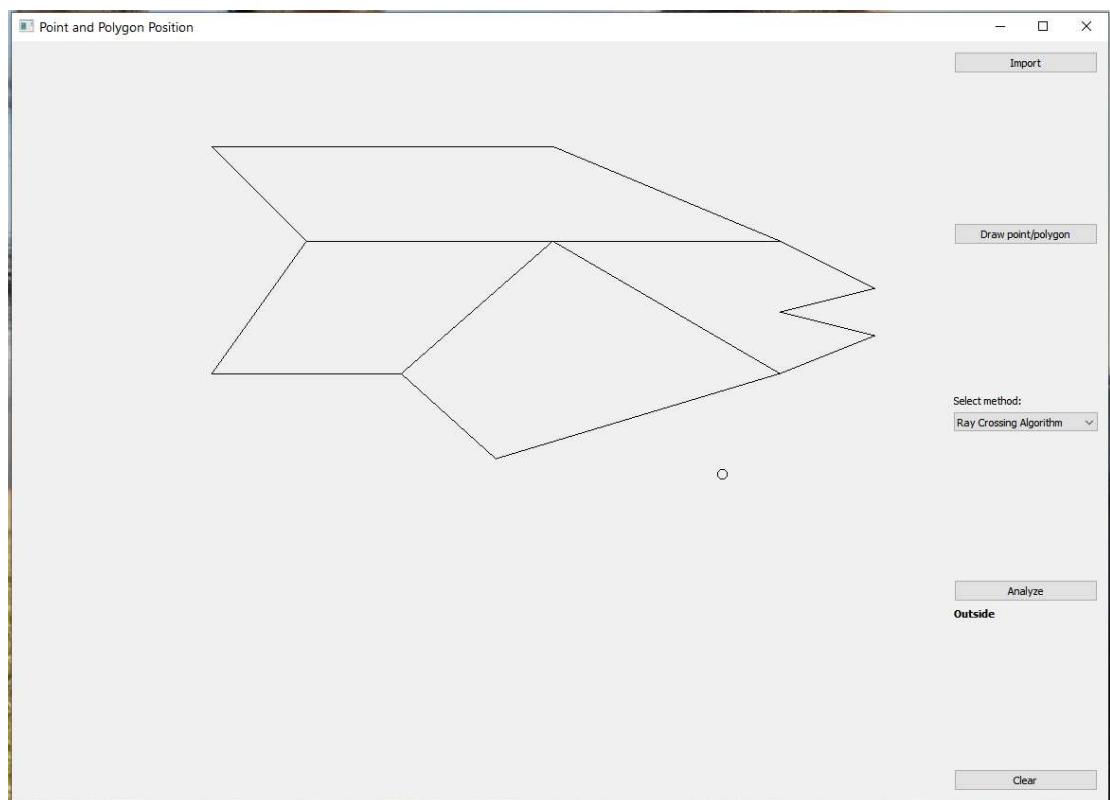
## 5. Printscreeny z vytvorenej aplikácie



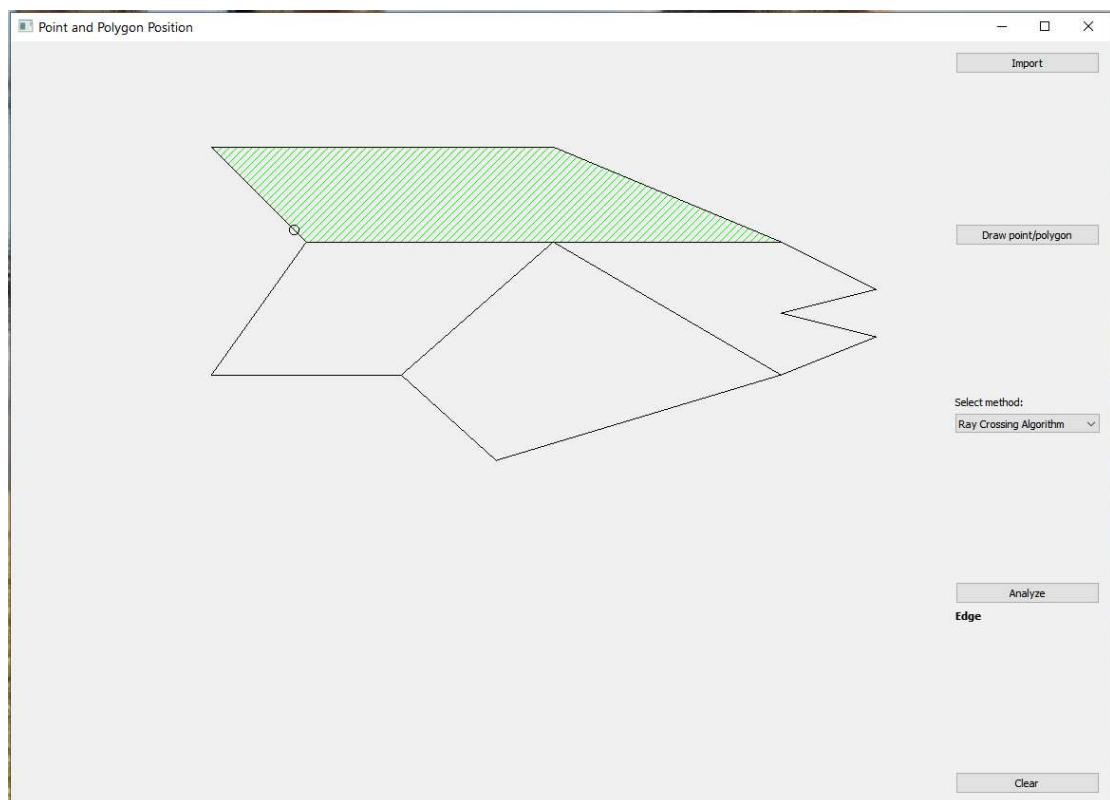
Obrázok 7 - Úvodné okno pri spustení aplikácie



Obrázok 8 - Vysvetlanie polygónu, v ktorom bod leží



Obrázok 9 - Výsledok, keď bod nenáleží polygónu



Obrázok 10 - Singulárny prípad - bod na hrane

## 6. Dokumentácia

### 6.1 Trieda algorithms.h

```
class Algorithms
{
public:
    Algorithms();
    // Funkcia počíta orientáciu body voči priamke a či sa bod nachádza v ľavej alebo pravej
    // polrovine. Do funkcie vstupujú súradnice bodu q a koncové body priamok.
    int getPointLinePosition(QPoint &q, QPoint &p1, QPoint &p2);
    // Funkcia na výpočet uhlu medzi dvoma priamkami danými dvoma vrcholmi polygónu pomocou
    // vektorov.
    double getAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4);
    // Funkcia algoritmu Winding Number. Vstupujú do nej súradnice bodu q a vrcholové body
    // polygónu.
    int getPositionWinding(QPoint &q, std::vector<QPoint> &pol);
    // Funkcia algoritmu Ray Crossing Algorithm. Vstupujú do nej súradnice bodu q a vrcholové
    // body polygónu.
    int getPositionRay(QPoint &q, std::vector<QPoint> &pol);
};
```

### 6.2 Trieda draw.h

```
class Draw : public QWidget
{
private:
    Q_OBJECT
    // Deklarácia kreslenia bodu alebo polygónu
    boolean draw_mode; //true = polygon, false = point
    // Deklarácia polygónu
    QPolygon polygon;
    // Deklarácia premennej na načítanie polygónov
    std::vector<QPolygon> polygons;
    // Deklarácia premennej vloženého bodu q
    QPoint q;
    // Deklarácia premennej na rozhodnutie, či bude polygón vyšrafovovaný
    std::vector<int> result;

public:
    // Okno kreslenia
    explicit Draw(QWidget *parent = nullptr);
    // Funkcia pre získanie súradníc bodov z Canvasu (čo sa stane po kliknutí myšou do
    // Canvasu)
    void mousePressEvent(QMouseEvent *e);
    // Funkcia na kreslenie bodu alebo polygónu (a jeho šrafovania)
    void paintEvent(QPaintEvent *e);
    void changeMode(){draw_mode = !draw_mode;}
    QPoint &getPoint(){return q;}
    // Funkcia na načítanie polygónov z textového súboru
    void loadFile(std::string &string_path);
    // Funkcia na získanie súradníc z polygónov
    std::vector<QPolygon> getPolygons (){return polygons;}
    // Funkcia na získanie výsledkov pre následné vyfarbenie polygónu
    void setResult(std::vector<int> res){result = res;}
    // Funkcia mazania grafického okna
    void clearC();
};
```

### 6.3 Trieda widget.h

```
class Widget : public QWidget
{
private:
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

private slots:
    // Čo sa stane kdeľ klikneme na ...
    void on_drawPointPolygon_clicked();
    void on_analyze_clicked();
    void on_clear_clicked();
    void on_importPolygons_clicked();
```

```

private:
    Ui::Widget *ui;
};



## 6.4 algorithms.cpp


int Algorithms::getPointLinePosition(QPoint &q, QPoint &p1, QPoint &p2) {
    //Tolerancia
    double eps = 1.0e-3;

    // Získanie bodovej a líniovej pozicie
    double ux = p2.x() - p1.x();
    double uy = p2.y() - p1.y();
    double vx = q.x() - p1.x();
    double vy = q.y() - p1.y();

    //Determinant
    double t = ux * vy - uy * vx;

    //qDebug() << t;

    // Násobok smerového vektora (p1p2) pre súradnice bodu "q"
    double m = vx / ux;

    //Bod v ľavej polrovine
    if (t > eps)
        return 1;

    // Bod v pravej polrovine
    if (t < -eps)
        return 0;

    // Bod na hrane
    if (t >= -eps and t <= eps and m >= 0 and m <= 1)
        return -1;

    // Bod na linii, ale nie na hrane
    return -2;
}

double Algorithms::getAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4) {
    //Získanie vektorov u, v
    double ux = p2.x() - p1.x();
    double uy = p2.y() - p1.y();
    double vx = p4.x() - p3.x();
    double vy = p4.y() - p3.y();

    //Normy vektorov
    double nu = sqrt(ux * ux + uy * uy);
    double nv = sqrt(vx * vx + vy * vy);

    //Skalárny súčin
    double dot = ux * vx + uy * vy;

    // Vrátenie uhlu omeša dvoch vektorov
    return fabsacos(dot / (nu * nv)));
}

int Algorithms::getPositionWinding(QPoint &q, std::vector<QPoint> &pol) {
    //Pozícia bodu a polygónu: Winding Number Algorithm
    //q v P: 1
    //q mimo P: 0
    //q na hrane P : -1

    double Omega = 0.0;

    //Nastavenie tolerancie
    const double eps = 1.0e-3;

    //Suma vrcholov polygónov
    const int n = pol.size();

    //Spracovanie všetkých vrcholov polygónov
    for (int i = 0; i < n; i++) {

        //Výpočet uhlu
        double om = getAngle(pol[i], q, pol[(i+1)%n], q);
    }
}

```

```

//Získanie orientácie bodu a hrany polygónu
int orientation = getPointLinePosition(q, pol[i], pol[(i+1)%n]);

// Bod v ľavej polrovine
if (orientation == 1)
    Omega += om;

// Bod v pravej polrovine
else
    Omega -= om;
//Bod v polygóne
if (fabs(fabs(Omega) - 2 * M_PI) <= eps)
    return 1;

//Bod mimo polygónu
else if (fabs(fabs(Omega)) <= eps) {
    return 0;
}

//Bod na hrane
else {
    return -1;
}

int Algorithms::getPositionRay(QPoint &q, std::vector<QPoint> &pol) {
    // Pozícia bodu a polygónu: Ray Crossing Algorithm
    //q v P: 1
    //q mimo P: 0

    //Počet priesečníkov v pravej polrovine
    int k = 0;

    // Počet priesečníkov v ľavej polrovine
    int l = 0;

    // Tolerancia
    double eps = 1.0e-3;

    //Suma vrcholov polygónov
    const int n = pol.size();

    // Spracovanie všetkých vrcholov polygónov
    for (int i = 1; i < n + 1 ; i++){
        //Výpočet redukovaných súradníci xi, yi, xii, yi
        double xir = pol[i%n].x() - q.x();
        double yir = pol[i%n].y() - q.y();
        double xiir = pol[i-1].x() - q.x();
        double yiir = pol[i-1].y() - q.y();

        //Pretina segment (pi-1, pi) ray paprsok ?
        if ((yir > 0) && (yiir <= 0) || (yiir > 0) && (yir <= 0)){
            //Výpočet x-ovej súradnice priesečníka
            double xm = (xir * yiir - xiir * yir)/(yir - yiir);

            //Bodt q v pravej polrovine
            if (xm > -eps)
                k += 1;

            // Bodt q v ľavej polrovine
            if (xm < eps)
                l += 1;
        }
    }

    // Bod na hrane
    if (k%2 > l%2 or k%2 < l%2)
        return -1;

    //Získanie poštu priesečníkov: párne alebo nepárne ?
    return k%2;
}

```

## 6.5 draw.cpp

```

Draw::Draw(QWidget *parent) : QWidget(parent){
    //Nastavenie počiatočného draw mode
    draw_mode = true;

    //Kreslenie bodu mimo Canvas
    q.setX(-50);

```

```

q.setY(-50);}

void Draw::mousePressEvent(QMouseEvent *e) {
    //Pozícia kurzora
    double x = e->pos().x();
    double y = e->pos().y();

    if (draw_mode){
        //Vytvorenie polygónu
        QPoint p(x, y);

        //Pridanie bodu do polygónu
        polygon.push_back(p);

        //Vytvorenie bodu
        else{
            //Zmena súradníc q
            q.setX(x);
            q.setY(y);}

        polygons.push_back(polygon);
        repaint();}

void Draw::paintEvent(QPaintEvent *e) {
    //Kreslenie
    QPainter painter(this);
    painter.begin(this);

    //Vytvorenie polygónu
    QPolygon pol;

    //Vloženie všetkých bodov do polygónu
    for (int i = 0; i < polygons.size(); i++ ){
        pol.append(polygons[i]);}

    //Kreslenie polygónu
    painter.drawPolygon(pol);

    //Nastavenie farby na vyšrafovanie polygónu
    QBrush brush;
    QPainterPath path;
    QPolygon colored_polygon;
    brush.setColor(Qt::green);
    brush.setStyle(Qt::BDiagPattern);

    //Vykreslenie polygónu obsahujúceho bod q
    for(int i = 0; i < result.size(); i++){
        if(result[i] == 1 || result[i] == -1){
            colored_polygon << polygons[i];
            path.addPolygon(colored_polygon);
            painter.fillPath(path, brush);
            painter.drawPolygon(colored_polygon);
            colored_polygon.clear();}}

    //Kreslenie bodu q
    int r = 5;
    painter.drawEllipse(q.x(), q.y(), 2 * r, 2 * r);

    //Koniec kreslenia
    painter.end();}

void Draw::loadFile(std::string &string_path) {
    polygons.clear();
    repaint();
    int id;
    double x;
    double y;
    QPolygon polygon;

    //Cesta k textovému súboru
    std::ifstream polygonFile(string_path);

    if(polygonFile.is_open()){
        //Vyplnenie vektorov
        while(polygonFile >> id >> x >> y){
            //Nový polygón

```

```

        if (id == 1){
            if (polygon.isEmpty() == FALSE) {
                polygons.push_back(polygon);
            }
            polygon.clear();
            polygon << QPoint(x, y);
        } else
            polygon << QPoint(x, y);

        //Pridanie posledného polygónu do vektoru
        polygons.push_back(polygon);

        //Zatvorenie súboru polygón
        polygonFile.close();

        //Prekreslenie okna
        repaint();
    }

void Draw::clearC(){
    //Vymazanie polygónov
    polygon.clear();
    polygons.clear();

    //Kreslenie bodu mimo Canvas
    q.setX(-50);
    q.setY(-50);

    //Odstránenie polygónov z Canvasu
    result.clear();

    //Prekreslenie okna
    repaint();
}

```

## 6.6 widget.cpp

```

void Widget::on_drawPointPolygon_clicked(){
    //Zmena kresliaceho módu
    ui->Canvas->changeMode();

void Widget::on_analyze_clicked(){
    //Získanie bodu q
    QPoint q= ui->Canvas->getPoint();

    //Získanie polygónu
    std::vector<QPolygon> polygons = ui->Canvas->getPolygons();

    //Získanie statusu combo-boxu, výber metódy
    Algorithms alg;
    QPolygon one_polygon;
    std::vector<int> result;
    std::vector<QPoint> pol;
    int p = 0;
    int res = 0;

    //Získanie polygónu
    for (int j = 0; j < polygons.size(); j++) {
        one_polygon = polygons[j];
        for (int i= 0; i< one_polygon.size(); i++) {
            pol.push_back(one_polygon[i]);
        }

        for (int k = 0; k < pol.size(); k++) {
            if (q == pol[k])
                p += 1;
        }
    }

    //Kontrola či je bod q na vrchole
    if (p < 1){
        //výber metódy
        if (ui->comboBox->currentIndex() == 0){
            res = alg.getPositionWinding(q, pol);
        }
        else
            res = alg.getPositionRay(q, pol);
        result.push_back(res);
    }
    else
        result.push_back(1);
    pol.clear();
    p=0;
}

//Popis, kde sa nachádza bod

```

```

int i = 1, j = -1, k = -2;
if (std::count(result.begin(), result.end(), i))
    ui->label->setText("Inside");
else if (std::count(result.begin(), result.end(), j))
    ui->label->setText("Edge");
else if (std::count(result.begin(), result.end(), k))
    ui->label->setText("Line");
else
    ui->label->setText("Outside");

//Uloženie premennej res do hodnoty result
ui->Canvas->setResult(result);

//Prekreslenie okna
ui->Canvas->repaint();}

void Widget::on_clear_clicked(){
//Vymazanie canvasu
ui->Canvas->clearC();
ui->label->setText("");}

void Widget::on_importPolygons_clicked(){
//Výber textového súboru v dialógu
QString path = QFileDialog::getOpenFileName(this, tr("Open Text File"), "../",
                                              tr("Text Files (*.txt)"));

//Vrátenie textu
std::string string_path = path.toStdString();

//Import polygónov
ui->Canvas->loadFile(string_path);}


```

## 7. Záver

Bola vytvorená funkčná aplikácia nevykazujúca žiadne chyby, ktoré by neumožňovali jej preloženie a spustenie. Bol vyriešený problém, ktorý nastal pri bode ležiacom na hrane polygónu. Singulárny prípad kde bod je vložený do vrcholu polygónu a vyznačenie singulárnych prípadov nebolo u aplikácií riešené. Tiež nie je navrhnutý algoritmus na automatické generovanie nekonvexných polygónov. Tieto bonusové úlohy by mohli byť ďalším zdokonalením našej aplikácie.

## 8. Zdroje

- [online]. Copyright © [cit. 22.10.2020]. Dostupné z: <https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk3.pdf>
- [online]. Copyright © [cit. 22.10.2020]. Dostupné z: [https://cloud2m.edupage.org/cloud/18\\_-\\_Analyticka\\_geometria%281%29.pdf?z%3Afcihigy9AJWKDvcihkxT4ZASMiHCB0kmemU7LRIQo7c%2Fu](https://cloud2m.edupage.org/cloud/18_-_Analyticka_geometria%281%29.pdf?z%3Afcihigy9AJWKDvcihkxT4ZASMiHCB0kmemU7LRIQo7c%2Fu)

## 9. Zoznam obrázkov

Obrázok 1 - Prevod na lokálnu procedúru.....	4
Obrázok 2 - Určovanie počtu priesčníkov.....	4
Obrázok 3 - Vedenie priamok CCW .....	5
Obrázok 4 - Určovanie Winding Number .....	5
Obrázok 5 - Určovanie počtu priesčníkov (Ray Crossing Alg.).....	6
Obrázok 6 - Rozdelenie polygónu na polroviny.....	6
Obrázok 7 - Úvodné okno pri spustení aplikácie .....	8
Obrázok 8 - Vysvetlanie polygónu, v ktorom bod leží.....	8
Obrázok 9 - Výsledok, keď bod nenáleží polygónu .....	9
Obrázok 10 - Singulárny prípad - bod na hrane .....	9