ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE, KARTOGRAFIE A GEOINFORMATIKA
KATEDRA SPECIÁLNÍ GEODÉZIE

název předmětu
ALGORITMY DIGITÁLNÍ KARTOGRAFIE A GIS
název úlohy
Úloha 2: Generalizace budov

akademický rok	semestr	vypracoval(a)	Datum	klasifikace
2023/24	Letní	Jan Koudelka, Vojtěch Müller	22.4.2024	

Technická zpráva

1 Zadání

Vytvořte aplikace, ve které bude možné načíst data ze souboru. Tyto data budou představována lomovými body budov. Do aplikace dále implementujte funkce, kterými bude možné budovy generalizovat do úrovně detailu LOD0. Pro tyto účely použijte vhodnou datovou sadu, např. data ze ZABAGED a testování proveďte nad třemi datovými sadami (historické centrum města, intravilán – sídliště, intravilán – izolovaná zástavba).

Generalizace bude provedena metodami:

- Minimum Area Enclosing Rectangle
- PCA
- Longest Edge
- Wall Average
- Weighted Bisector

U první metody použijte algoritmus Jarvis Scan pro konstrukci konvexní obálky. Budovu při generalizaci do úrovně LOD0 nahraďte obdélníkem orientovaným v obou hlavních směrech, se středem v těžišti budovy, jeho plocha bude stejná jako plocha budovy. Výsledky generalizace vhodně vizualizujte.

Otestujte a porovnejte efektivitu všech metod s využitím hodnotících kritérií. Pokuste se rozhodnout, pro které tvary budov dávají metody nevhodné výsledky, a pro které naopak poskytují vhodnou aproximaci.

2 Kroky

V rámci úlohy byly zpracovány tyto části:

- Generalizace budov metodami Minimum Area Enclosing Rectangle a PCA. – 15 bodů
- Generalizace budov metodou Longest Edge. – 5 bodů
- Generalizace budov metodou Wall Average. – 8 bodů
- Generalizace budov metodou Weighted Bisector. – 10 bodů

3 Generalizace budov

Generalizace budov je problematika, která se zabývá zjednodušením tvaru budovy, například za účelem zobrazení v určitém přiblížení, které by mohlo být v komplikovaných tvarech budov nepřehledné.

3.1 Konstrukce konvexní obálky

Důležitým algoritmem pro generalizaci budov je konstrukce konvexní obálky. Jedna z definic konvexní obálky je: *„Konvexní obálka H konečné množiny n bodů P představuje konvexní mnohoúhelník S s nejmenší plochou.“* [1]. Jedná se tedy o polygon, který obsahuje všechny body ze zadané množiny a zároveň má nejmenší možnou plochu. Pro konstrukci konvexní obálky existuje několik algoritmů.

3.1.1 Jarvis Scan

Tento algoritmus pro tvorbu konvexní obálky připomíná postup balení dárku do papíru, a proto se mu také říká Gift Wrapping Algorithm. Předpokladem k použitelnosti tohoto algoritmu je, že množina bodů P neobsahuje tři kolineární body.

Principem algoritmu je, že k aktuálním posledním dvěma bodům konvexní obálky hledáme nový bod, který maximalizuje úhel. Tento bod je poté přidán do obálky. Je ovšem nutno nejprve nalézt výchozí úsečku, od které se následné přidávání bodů provádí.

Algoritmus 1: Jarvis Scan

```
def createCH(self, pol:QPolygonF):
        #Create Convex Hull using Jarvis Scan
        ch = QPolygonF()

        #Find pivot q (minimize y)
        q = min(pol, key = lambda k: k.y())

        #Find left-most point (minimize x)
        s = min(pol, key = lambda k: k.x())

        #Initial segment
        pj = q
        pj1 = QPointF(s.x(), q.y())

        #Add to CH
        ch.append(pj)

        # Find all points of CH
        while True:
            #Maximum and its index
            omega_max = 0
            index_max = -1

            #Browse all points
            for i in range(len(pol)):

                if pj != pol[i]:

                    #Compute omega
```

```
                omega = self.get2VectorsAngle(pj, pj1, pj, pol[i])

                #Actualize maximum
                if(omega>omega_max):
                    omega_max = omega
                    index_max = i

        #Add point to the convex hull
        ch.append(pol[index_max])

        #Reasign points
        pj1 = pj
        pj = pol[index_max]

        # Stopping condition
        if pj == q:
            break

    return ch
```

V algoritmu je nejprve nalezen pivot $q, q = \min(y_i)$, tedy existující bod s minimální souřadnicí y. Dále je vytvořen nový bod, který má minimalizované obě souřadnice. Tyto body tvoří inicializační segment, ke kterému je dále prováděno hledání konvexní obálky. Pivot je prvním bodem, který je do obálky přidán. Následně je vyhledáván úhel ω pro jednotlivé body. Po nalezení maximálního úhlu ω je bod s tímto úhlem přidán do konvexní obálky a linie, ke které je hledaný nový úhel je předefinována na pivot, tedy předposlední přidaný bod a poslední přidaný bod. Takto probíhá hledání bodů v cyklu *while* dokud není nový bod roven pivotu.

3.2 Vytvoření min-max boxu

Min-max box je obdélník totožný se základními osami, který ohraničuje ostatní prvky. Tento obdélník je vytvořen na základě nalezení minimálních a maximálních souřadnic všech prvku a tyto souřadnice tvoří hrany obdélníka.

Algoritmus 2: Min-max box

```
def createMMB(self, pol:list[QPolygonF]):
        # Create min max box and compute its area

        #Points with extreme coordinates
        p_xmin = min(pol, key = lambda k: k.x())
        p_xmax = max(pol, key = lambda k: k.x())
        p_ymin = min(pol, key = lambda k: k.y())
        p_ymax = max(pol, key = lambda k: k.y())

        #Create vertices
        v1 = QPointF(p_xmin.x(), p_ymin.y())
        v2 = QPointF(p_xmax.x(), p_ymin.y())
        v3 = QPointF(p_xmax.x(), p_ymax.y())
        v4 = QPointF(p_xmin.x(), p_ymax.y())

        #Create new polygon
        mmb = QPolygonF([v1, v2, v3, v4])

        #Area of MMB
        area = (v2.x() - v1.x()) * (v3.y() - v2.y())

        return mmb, area
```

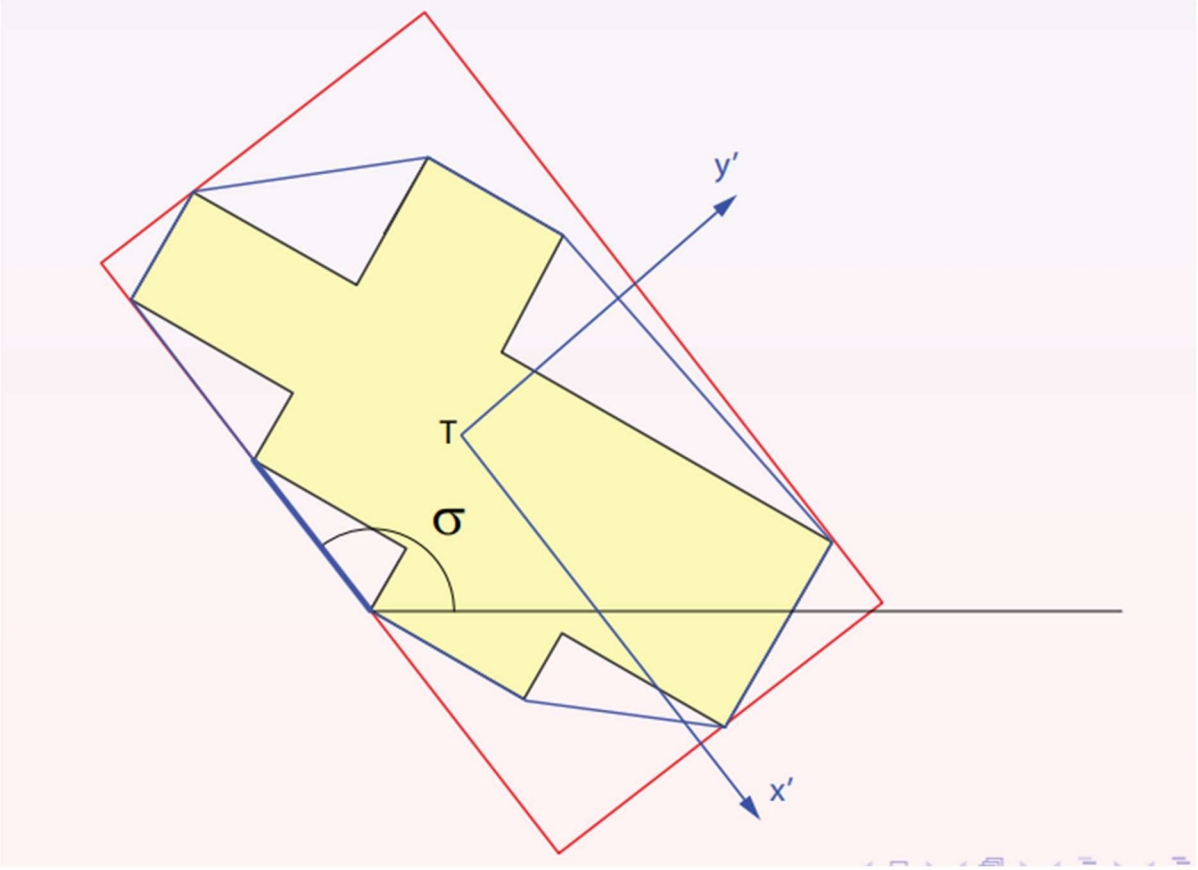
V algoritmu jsou nejprve nalezeny minimální a maximální souřadnice polygonu a následně vytvořeny čtyři vrcholy, jejichž souřadnice jsou kombinace těchto extrémních hodnot. Z těchto vrcholů je nakonec vytvořen polygon.

3.3 Detekce natočení budovy

V této úloze bude generalizace budovy provedena vytvořením obdélníku, orientovaného do dvou hlavních směrů. K tomuto zobrazení je potřeba implementovat algoritmy k detekci natočení budovy. V této úloze bylo použito několik těchto algoritmů a ty zde budou popsány.

3.3.1 Minimum Area Enclosing Rectangle

V tomto algoritmu hledáme obdélník s minimální plochou, pro který platí, že alespoň jedna strana je kolineární s konvexní obálkou zadaných bodů. Myšlenka je založena na opakované rotaci množiny bodů a jejich konvexní obálky a následném vytváření min-max boxů pro otočenou množinu. Z těchto boxů je pak vybrán ten s minimální plochou.



Obrázek 1: Minimum Area Enclosing Rectangle [1]

Algoritmus 3: Minimum Area Enclosing Rectangle

```
def createMBR(self, pol: list[QPolygonF]):
        # Create minimum area enclosing rectangle
        er_r_list = []

        for j in pol:

            #Create convex hull
            ch = self.createCH(j)

            #Get minmax box, area and sigma
            mmb_min, area_min = self.createMMB(ch)
            sigma_min = 0

            # Process all segments of ch
            for i in range(len(ch)-1):
                # Compute sigma
                dx = ch[i+1].x() - ch[i].x()
                dy = ch[i+1].y() - ch[i].y()
                sigma = atan2(dy,dx)

                #Rotate convex hull by sigma
                ch_rot = self.rotatePolygon(ch, -sigma)

                #Find min-max box over rotated ch
                mmb, area = self.createMMB(ch_rot)
```

```
            #Actualize minimum area
            if area < area_min:
                area_min = area
                mmb_min = mmb
                sigma_min = sigma

        #Rotate minmax box
        er = self.rotatePolygon(mmb_min, sigma_min)




        #Resize rectangle
        er_r = self.resizeRectangle(er, j)
        er_r_list.append(er_r)

    return er_r_list
```

V algoritmu je nejprve vytvořena konvexní obálka ch. Dále je inicializován min-max box pro neotočenou množinu a ten je zvolen za obdélník s aktuálně minimální plochou. Také je inicializován minimální úhel σ.

Následně přes *for* cyklus probíhá procházení jednotlivých segmentu konvexní obálky. Je vypočten úhel σ pro natočení konvexní obálky tak, aby byl daný segment rovnoběžný s hlavní osou. Nyní je konvexní obálka otočena právě o určený úhel σ a dále je vytvořen min-max box. Ze všech vytvořených min-max boxů je poté vybrán ten s minimální plochou. Nakonec proběhne zpětná rotace vybraného obdélníka

3.3.2 PCA

Tato metoda používá k určení hlavních směrů singulární rozklad (SVD).

Kovarianční matic:

$$C = \begin{bmatrix} C(A,A) & C(A,B) \\ C(B,A) & C(B,B) \end{bmatrix}, C(A,B) = \frac{1}{n-1}\sum_{i=1}^{n}(A_i - \mu_A)(B_i - \mu_B)$$

Singulární rozklad:

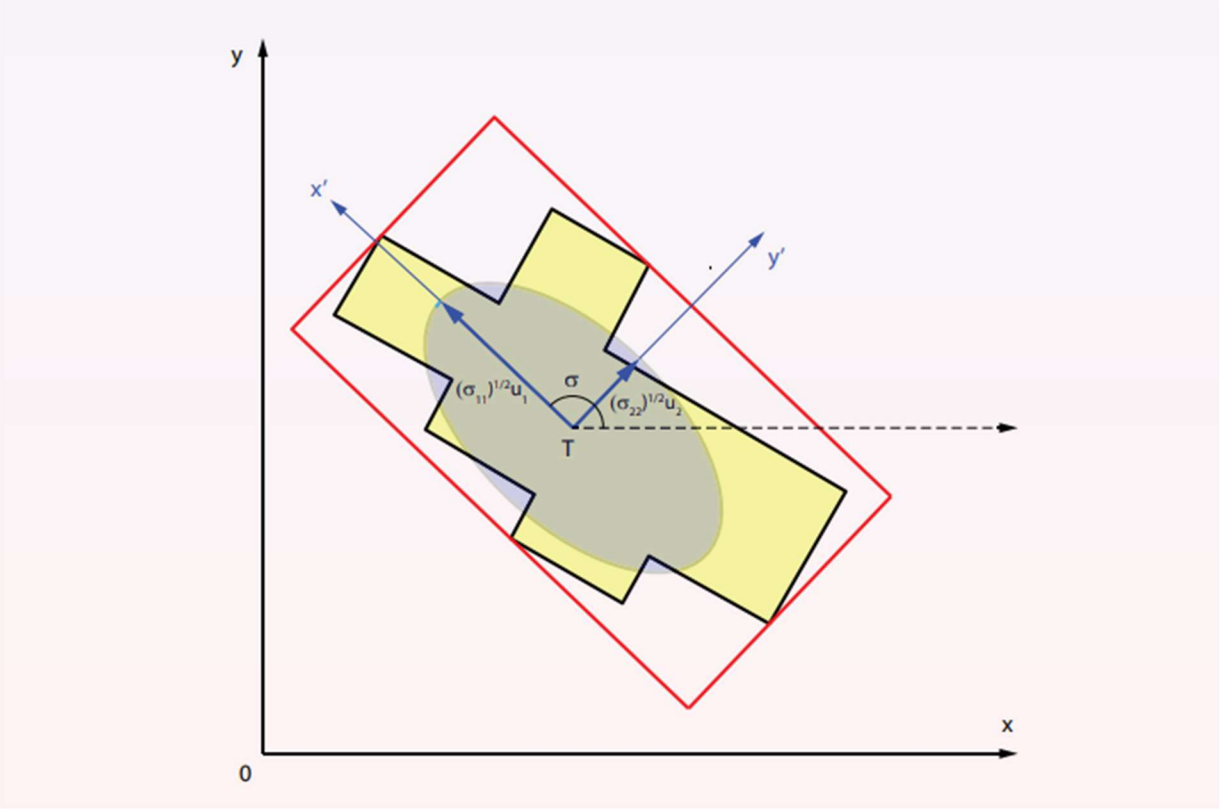
$$C = U\Sigma V^T, \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix}^T$$

$$U = V = \begin{bmatrix} \cos\sigma & -\sin\sigma \\ \sin\sigma & \cos\sigma \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix}$$

Rotace množiny P o úhel ± ω

$$P_0 = PV, P = V^{-1}P_0$$



Obrázek 2: PCA [1]

Algoritmus 4: PCA

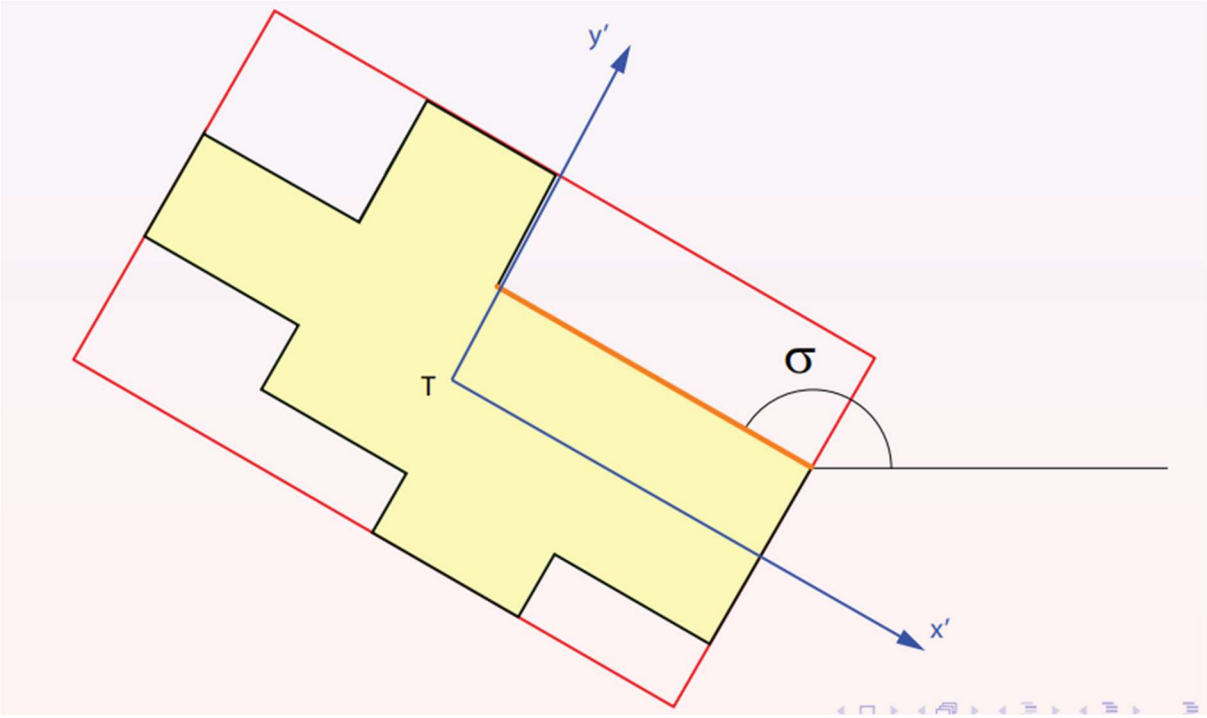
```
def createER_PCA(self, pol: QPolygonF):
        #create enclosing rectangle using PCA

        mmb_r_list = []

        for j in pol:

            x = []
            y = []

            #add x,y coordinates to the list
            for p in j:
                x.append(p.x())
                y.append(p.y())

            #convert to matrix
            A = array([x,y])

            #covariance matrix
            C = cov(A)

            #singular value decomposition
            [U,S,V] = svd(C)

            #compute sigma
            sigma = atan2(V[0][1], V[0][0])
```

```
            #Rotate polygon
            pol_rot = self.rotatePolygon(j, -sigma)

            #Find min-max box over rotated polygon
            mmb, area = self.createMMB(pol_rot)

            #Rotate minmax box
            mmb_rot = self.rotatePolygon(mmb, sigma)

            #Resize rectangle
            mmb_r = self.resizeRectangle(mmb_rot, j)

            mmb_r_list.append(mmb_r)

        return mmb_r_list
```

V algoritmu je nejprve vytvořena matice ze všech souřadnic x a y polygonu. Následně je vytvořena kovariantní matice a pomocí singulárního rozkladu určena matice V. Z této matice je poté pomocí funkce tangens vypočítán úhel σ. O tento úhel poté proběhne otočení množiny, následně bude vytvořen min-max box a poté bude zpětnou rotací získán výsledný polygon.

3.3.3 Longest Edge

V této metodě je hlavní směr budovy představován nejdelší stranou budovy.



Obrázek 3: Longest Ege [1]

Algoritmus 5: Longest Edge

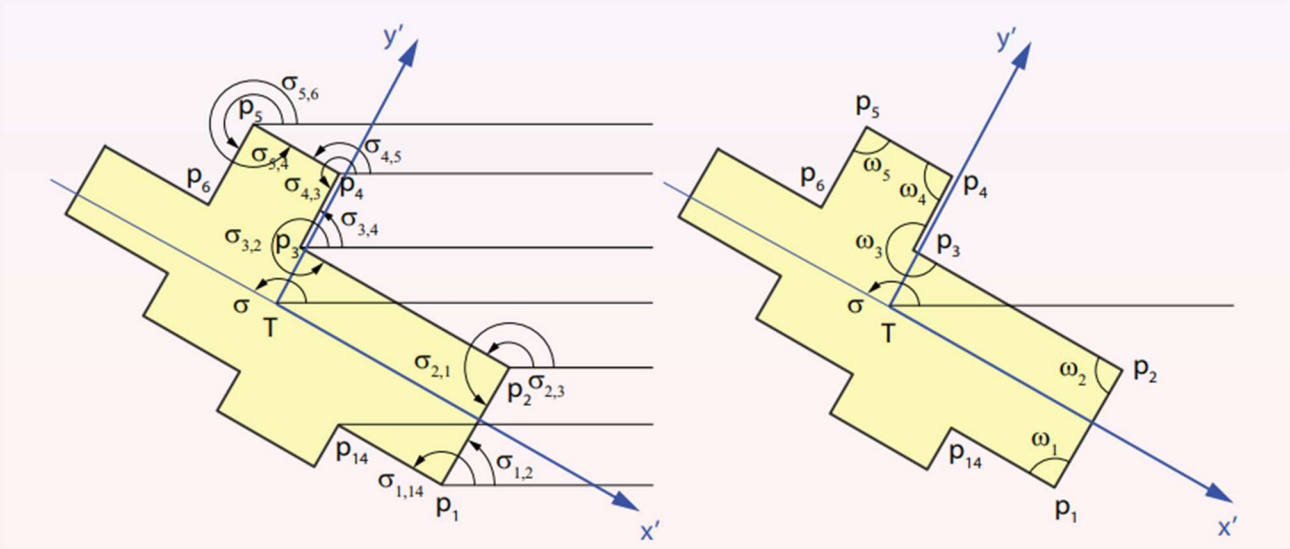
```
def createER_LongestEdge(self, pol: QPolygonF):
        #create enclosing rectangle using Longest Edge

        mmb_r_list = []

        for j in pol:
            #find longest edge
            max_dist = 0

            for i in range(len(j)-1):
                # Compute distance and sigma
                dx = j[i+1].x() - j[i].x()
                dy = j[i+1].y() - j[i].y()
                d = sqrt(dx*dx+dy*dy)
                if (d>max_dist):
                    max_dist = d
                    sigma = atan2(dy,dx)

            #Rotate polygon
            pol_rot = self.rotatePolygon(j, -sigma)

            #Find min-max box over rotated polygon
            mmb, area = self.createMMB(pol_rot)

            #Rotate minmax box
            mmb_rot = self.rotatePolygon(mmb, sigma)

            #Resize rectangle
            mmb_r = self.resizeRectangle(mmb_rot, j)

            mmb_r_list.append(mmb_r)

        return mmb_r_list
```

V algoritmu je nejprve inicializována maximální délka, která je určena na 0. Poté jsou procházeny jednotlivé strany budovy a hledána ta, která je nejdelší. K nejdelší straně je pak ukládán úhel k otočení, aby byla tato strana rovnoběžná s hlavní osou. Nakonec proběhne rotace budovy o tento úhel, vytvoření min-max boxu a zpětná rotace.

3.3.4 Wall Average

V této metodě je na vnitřní úhel u jednotlivých vrcholů aplikována operace $mod\left(\frac{\pi}{2}\right)$, tedy u každého vrcholu je určen přesah přes nejbližší čtvrtinu kruhu. Tato hodnota je poté pro hodnoty větší než $\frac{\pi}{4}$ převedena do záporných hodnot, tedy úhel se poté pohybuje v intervalu $\left(-\frac{\pi}{4}, \frac{\pi}{4}\right)$. Ze všech úhlů je poté vypočten vážený průměr ze vzorce

$$\sigma = \sigma_{1,2} + \frac{\sum_{i=1}^{n} r_i s_i}{\sum_{i=1}^{n} s_i}$$



Obrázek 4: Wall Average [1]

Algoritmus 6: Wall Average

```
def createER_WallAverage(self, pol: QPolygonF):
        #create enclosing rectangle using Wall Average

        mmb_r_list = []
        effi_list = []

        for j in pol:

            #compute sigma
            n = len(j)
            sumaA = 0
            sumaB = 0
            for i in range(n):
                #condition replacing modulo
                if (i!= (n-1) and i != 0):
                    a = i
                    b = i+1
                    c = i-1
                #connection between the first and the last point of the polygon
                elif (i==0):
                    a = i
                    b = i+1
                    c = n-1
                #connection between the last and the first point of the polygon
                else:
                    a = i
                    b = 0
                    c = i-1

                # Compute distance and sigmas
                dx1 = j[c].x() - j[a].x()
                dy1 = j[c].y() - j[a].y()
                sigma1 = atan2(dy1,dx1)
```

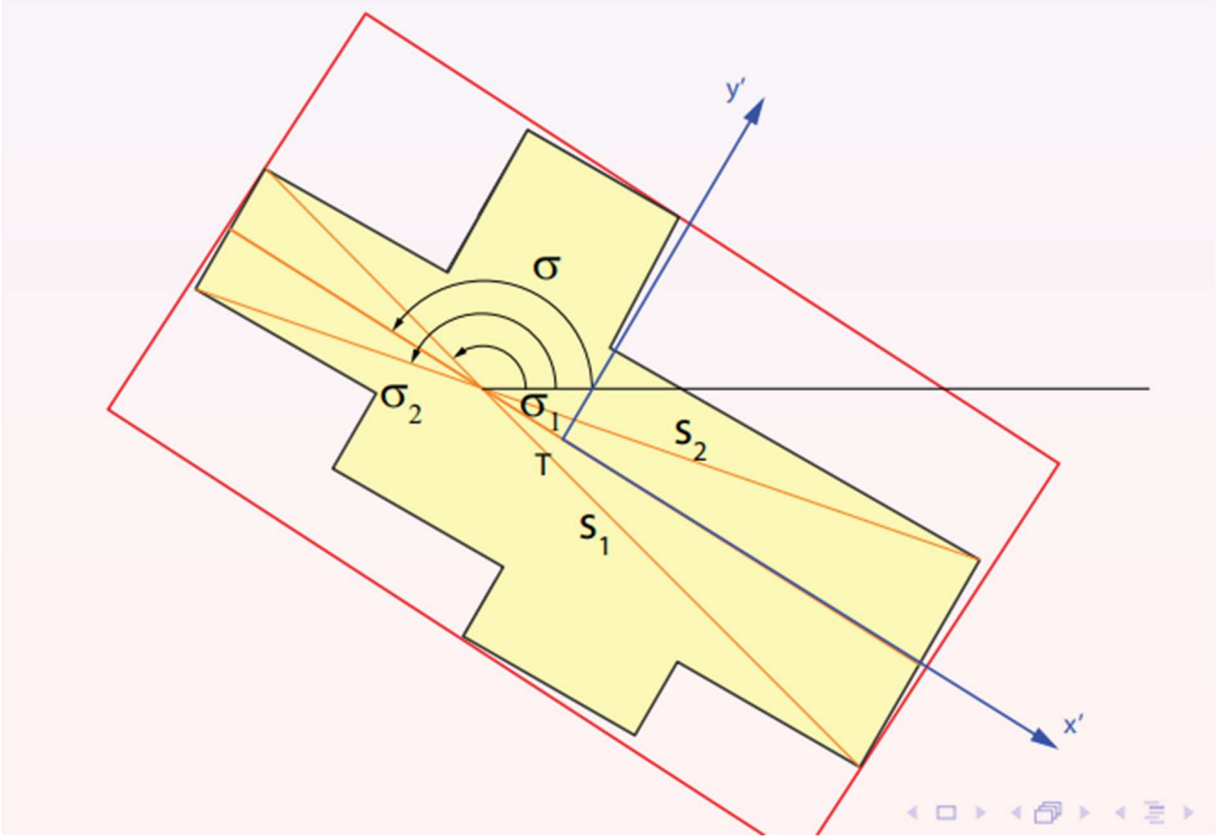
```
            dx2 = j[b].x() - j[a].x()
            dy2 = j[b].y() - j[a].y()
            sigma2 = atan2(dy2,dx2)
            d = sqrt(dx2*dx2+dy2*dy2)

            #compute w
            w = sigma1-sigma2

            #compute residua
            k = 2*w/pi
            r = (k-floor(k))*(pi/2)
            if (w%(pi/2)<(pi/4)):
                r = r
            else:
                r = pi/2 - r

            sumaA = sumaA+r*d
            sumaB = sumaB+d

        #sigma12
        dx = j[1].x() - j[0].x()
        dy = j[1].y() - j[0].y()
        sigma12 = atan2(dy,dx)

        #final sigma
        sigma = sigma12 + sumaA/sumaB

        #Rotate polygon
        pol_rot = self.rotatePolygon(j, -sigma)

        #Find min-max box over rotated polygon
        mmb, area = self.createMMB(pol_rot)

        #Rotate minmax box
        mmb_rot = self.rotatePolygon(mmb, sigma)

        #Resize rectangle
        mmb_r = self.resizeRectangle(mmb_rot, j)
        effi = self.efficiency(mmb_r, j)

        mmb_r_list.append(mmb_r)
        effi_list.append(effi)

    return mmb_r_list, effi_list
```

V algoritmu probíhá nejprve výpočet směrníků a délek jednotlivých stran polygonu a následně výpočet vnitřního úhlu. Na tento úhel je poté aplikována výše zmíněná operace a ze všech úhlů je určen finální úhel σ. O tento úhel je poté provedena rotace s určením min-max boxu a následně je provedena zpětná rotace.

3.3.5 Weighted Bisector

V této metodě jsou hledány dvě nejdelší úhlopříčky. Z nich jsou zjištěny směrnice a délky a váženým průměrem je poté určen hlavní směr.

$$\sigma = \frac{s_1\sigma_1 + s_2\sigma_2}{s_1 + s_2}$$



Obrázek 5: Weighted Bisector [1]

Algoritmus 7: Weighted Bisector

```
def createER_WeightedBisector(self, pol: QPolygonF):
        #create enclosing rectangle using weighted bisector

        mmb_r_list = []
        effi_list = []

        for k in pol:

            #find 2 longest diagonalls
            max_diag_1 = 0
            max_diag_2 = 0
            i_h1 = j_h1 = i_h2 = j_h2 = -1
            x1i = x1j = y1i = y1j = x2i = x2j = y2i = y2j = 0

            for i in range(len(k)):
                for j in range(len(k)):
                    if (i==i_h1 and j==j_h1):
                        continue
```

```
                    # Compute diagonals
                    dx = k[j].x() - k[i].x()
                    dy = k[j].y() - k[i].y()
                    d = sqrt(dx*dx+dy*dy)
                    if (d>max_diag_1):
                        max_diag_1 = d
                        x1i = k[i].x()
                        y1i = k[i].y()
                        x1j = k[j].x()
                        y1j = k[j].y()
                        i_h1 = j
                        j_h1 = i

            i_h2 = j_h2 = -1
            for i in range(len(k)):
                for j in range(len(k)):
                    if (i==i_h2 and j==j_h2 or i==i_h1 or i==j_h1 or j==i_h1 or
j==j_h1):
                        continue

                    # Compute diagonals
                    dx = k[j].x() - k[i].x()
                    dy = k[j].y() - k[i].y()
                    d = sqrt(dx*dx+dy*dy)
                    if (d>max_diag_2):
                        max_diag_2 = d
                        x2i = k[i].x()
                        y2i = k[i].y()
                        x2j = k[j].x()
                        y2j = k[j].y()
                        i_h2 = j
                        j_h2 = i



            x = ((x1i*y1j-y1i*x1j)*(x2i-x2j)-(x1i-x1j)*(x2i*y2j-y2i*x2j)) / ((x1i-
x1j)*(y2i-y2j)-(y1i-y1j)*(x2i-x2j))
            y = ((x1i*y1j-y1i*x1j)*(y2i-y2j)-(y1i-y1j)*(x2i*y2j-y2i*x2j)) / ((x1i-
x1j)*(y2i-y2j)-(y1i-y1j)*(x2i-x2j))


            #compute sigmas
            dx1 = x1i-x
            dy1 = y1i-y
            dx2 = x2i-x
            dy2 = y2i-y
            sigma1 = atan2(dy1,dx1)
            if (sigma1<0):
                sigma1 = sigma1+pi
            sigma2 = atan2(dy2,dx2)
```

```
            if (sigma2<0):
                sigma2 = sigma2+pi
            #sigma
            sigma = (sigma1*max_diag_1+sigma2*max_diag_2)/(max_diag_1+max_diag_2)

            #Rotate polygon
            pol_rot = self.rotatePolygon(k, -sigma)

            #Find min-max box over rotated polygon
            mmb, area = self.createMMB(pol_rot)

            #Rotate minmax box
            mmb_rot = self.rotatePolygon(mmb, sigma)

            #Resize rectangle
            mmb_r = self.resizeRectangle(mmb_rot, k)
            effi = self.efficiency(mmb_r, k)

            mmb_r_list.append(mmb_r)
            effi_list.append(effi)

        return mmb_r_list, effi_list
```

V algoritmu jsou nejprve inicializovány proměnné pro maximální úhlopříčky. Následně jsou procházeny všechny dvojice bodů a pokud se nenachází vedle sebe, tak se jedná o úhlopříčku. Z nich jsou poté vybrány ty dvě nejdelší a jejich směrnice a délky. Druhá nejdelší uhlopříčka nesměla vycházet z bodů nejdelší úhlopříčky. Z těchto hodnot je poté pomocí vzorce uvedeného výše určena hodnota σ a o tuto hodnotu je provedena rotace, následně vytvořen min-max box a poté provedena zpětná rotace.

3.4 Hodnocení efektivity metody

Pro zhodnocení efektivity byly použito hodnotící kritérium střední hodnoty úhlových odchylek jednotlivých segmentů. Pro určení byly použity tyto vzorce

$$\Delta\sigma_1 = \frac{\pi}{2n}\sum_{i=1}^{n}[r_i - r]$$

$$r_i = (k_i - \lfloor k_i \rfloor)\frac{\pi}{2}$$

$$k_i = \frac{2\sigma_i}{\pi}$$

Algoritmus 8: Hodnocení efektivity

```
def efficiency (self, mbr: QPolygonF, pol: QPolygonF):
        #computing efficiency of aproximation

        #compute main sigma of mbr
        d1 = sqrt((mbr[0].x()-mbr[1].x())**2 + (mbr[0].y()-mbr[1].y())**2)
        d2 = sqrt((mbr[1].x()-mbr[2].x())**2 + (mbr[1].y()-mbr[2].y())**2)
        if (d1>d2):
```

```
        sigma_mbr = atan2((mbr[0].y()-mbr[1].y()),(mbr[0].x()-mbr[1].x()))
    else:
        sigma_mbr = atan2((mbr[1].y()-mbr[2].y()),(mbr[1].x()-mbr[2].x()))

    #compute r of mbr
    k_mbr = 2*sigma_mbr/pi
    r_mbr = (k_mbr-floor(k_mbr))*(pi/2)

    #compute suma of residuas
    suma = 0
    n = len(pol)
    for i in range(n):
        #condition replacing modulo
        if (i != (n-1) and i != 0):
            a = i
            b = i+1
            c = i-1
        elif (i==0): #connection between the first and the last point of the polygon
            a = i
            b = i+1
            c = n-1
        else: #connection between the last and the first point of the polygon
            a = i
            b = 0
            c = i-1

        #sigma
        sigma = atan2((pol[b].y()-pol[a].y()),(pol[b].x()-pol[a].x()))

        #r
        k = 2*sigma/pi
        r = (k-floor(k))*(pi/2)

        #suma of residuas
        suma = suma + (r-r_mbr)**2

    dsigma = (pi/(2*len(pol))*sqrt(suma))*(180/pi) #in degrees
    return dsigma
```

V algoritmu je nejprve vypočten směrník generalizovaného obdélníka a z něj vypočtena hodnota r. Poté je stejný výpočet proveden pro jednotlivé body v polygonu budovy a je napočítávána suma z hodnot r. Podle výše uvedeného vzorce je nakonec vypočítána hodnota $\Delta\sigma_1$.

4 Problematické situace

Základní vytvoření aplikace probíhalo bez větších komplikací. Vytvoření prostředí, které zde probíhalo bylo obdobné jako v první úloze. Největší náročnost úlohy probíhala ve zpracování samotných algoritmů. V metodách Minimum Area Enclosing Rectangle, PCA a Longest Edge nevznikl žádný problém.

Při řešení metody Wall Average vzniklo určité množství problémů při psaní algoritmu. Vzorečky, které vychází ze zdroje jedna [1] obsahují některé části nejistě popsané a při provádění algoritmu musela být nakonec provedena hluboká analýza problému.

Metoda Weighted Bisector přinesla hlavní problém při hledání úhlopříček. Nejprve byly některé budovy s komplikovanějším tvarem hledány jednoznačně špatně. Po analýze byla nakonec přidána podmínka, při které nemohly dvě nejdelší úhlopříčky vycházet ze stejného bodu.

5 Vstupní data

Pro vstup do aplikace byly vytvořeny tři shapefile soubory. Jedná se o soubory s polygony jednotlivých budov. Data do těchto souborů byla vytvořena z dat ZABAGED, a to pomocí softwaru ArcGIS Pro, kde byla nahrána vrstva budov celé republiky a následně oříznuta na požadované území. Všechna území se nachází v Praze, konkrétně se jedná o centrum Prahy ve Vršovicích, sídliště na Chodově a izolovanou zástavbu také na Chodově. Všechna vstupní data se nachází v přílohách.

6 Výstupní data

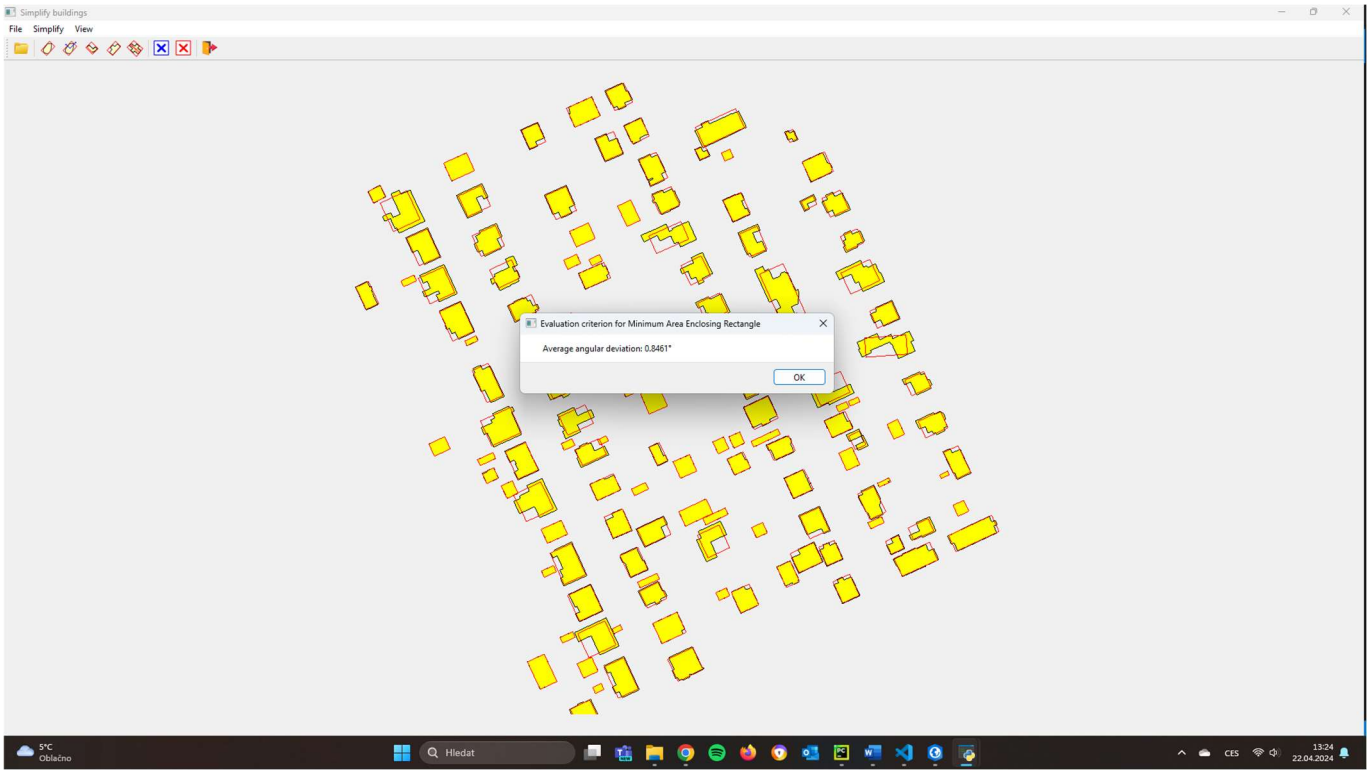
Výstupem úlohy je aplikace, která umožní nahrát shapefile soubor s polygony budov a následně pomocí různých tlačítek provádí pět metod generalizace budov. V prostředí aplikace jsou zobrazovány jak nahrané budovy, tak obdélníky generalizovaných budov. Aplikace při provedení generalizace také provede výpočet efektivity metody pomocí hodnotících kritérií. Tato hodnota je zobrazena ve vyskakovacím okně.

Výstupem je také zhodnocení jednotlivých metod pro různá vstupní data pomocí hodnotícího kritéria.

Tabulka 1: Hodnotící kritérium ve ° pro různé typy budov.

	MAER	PCA	Longest Edge	Wall Average	Weighted Bisector
Historické centrum města	3,9957	8,1300	3,8805	3,8801	5,8451
Intravilán – sídliště	1,0791	2,6482	0,8886	0,9428	1,9747
Intravilán – izolovaná zástavba	0,8461	7,9096	0,8128	1,0615	2,8263

7 Výsledná aplikace



Obrázek 6: Výsledná aplikace

8 Dokumentace

Aplikace při otevření zobrazí okno s prostorem pro vstup a výstup a v horní části se nachází lišta s prvky, které je možné použít. Pod lištou se také nachází Toolbar s jednotlivými možnostmi.

Lišta se skládá ze tří částí.

- File
- Simplify
- View

V části File se poté nachází metoda Open, která umožní nahrát vstup shapefilu do aplikace a metoda Exit, kterou lze aplikaci zavřít.

V části Simplify se poté nachází jednotlivé algoritmy pro výpočet generalizace budovy a to konkrétně

- Minimum Area Enclosing Rectangle
- PCA
- Longest Edge
- Wall Average
- Weighted Bisector

V části View se nachází metoda Clear results, která ze zobrazení vymaže polygony vytvořené generalizací budovy a metoda Clear All, která ze zobrazení odstraní všechny polygony a plochu vyčistí kompletně.

9 Závěr

Byla vytvořena aplikace, která umožňuje provést generalizaci budov různými metodami. Metody, které lze v rámci aplikace použít jsou Minimum Area Enclosing Rectangle, PCA, Longest Edge, Wall Average a Weighted Bisector.

Pro další výstup byly vytvořeny shapefile soubory s třemi typy budov. Těmito typy jsou historické centrum města, intravilán – sídliště a intravilán – izolovaná zástavba.

Pro zhodnocení různých metod pro různý typ zástavby bylo do aplikace přidáno vyskakovací okno, které zobrazí průměrné hodnotící kritérium pro budovy v daném souboru. Z těchto hodnot lze vidět, že budovy v historickém centru města mají hodnoty výrazně horší. Navíc při vizuálním zhodnocení výstupu lze vidět, že budovy v shapefilu nejsou vhodně zformovány pro použití klasické generalizace, která se v této aplikaci nachází. To hlavně z důvodu rozštěpení na sebe navazujících budov a z toho důvodu vznikajících přesahů mezi generalizovanými polygony.

Metoda PCA ve výsledných hodnotách vytváří výraznou odchylku oproti ostatním metodám a dává tedy horší výsledky. Metoda Wighted Bisector dává výsledky o něco lepší, ale také se odchylují od ostatních tří metod.

Z metody Longest Edge vychází podobné hodnotící kritérium po obě varianty intravilánové zástavby. Obě hodnoty jsou navíc poměrně nízké a metoda je tedy celkem vhodná, zřejmě z důvodu jednoduchosti tvarů budov.

Metoda Minimum Area Enclosing Rectangle je poté nižší u izolované zástavby a metoda Wall Average naopak u sídliště. Toto je zřejmě z důvodu protažení budov na sídlišti. Pro tyto dva typy zástavby jsou tedy vhodné příslušné metody.

Pro různé metody tedy existují vhodné tvary budov. Pro metodu MAER je zřejmě vhodnější pravidel osově symetrický tvar, nejlépe jednoduchý čtverec. Metoda Wall Average funguje vhodně u protaženějších budov a metoda Longest Edge u budov, které mají hlavní stranu delší než ostatní. Metoda Weigted Bisectro je poté podobně jako MAER vhodná u osově symetrických budov. Metoda PCA nejlépe funguje u budov bez výraznějších výstupků.

10 Přílohy

- Skript pro spuštění aplikace (*MainForm.py*)
- Skript pro vykreslování do aplikace (*draw.py*)
- Skript pro výpočet algoritmů (*algorithms.py*)
- Skript pro načtení dat (*pio.py*)
- Složka s ikonami (*images*)
- Složka se vstupními daty (*polygons*)

11 Zdroje

[1] *https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk4_new.pdf*

V Praze dne 22.4.2024 **Jan Koudelka, Vojtěch Müller**