HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Modul G386 – Kartographie und Geoprocessing

Diskussion von Varianten zur Formvereinfachung von Gebäudegrundrissen

(Windungsgeneralisierung u.a. nach Mindestlängen, Flächen)

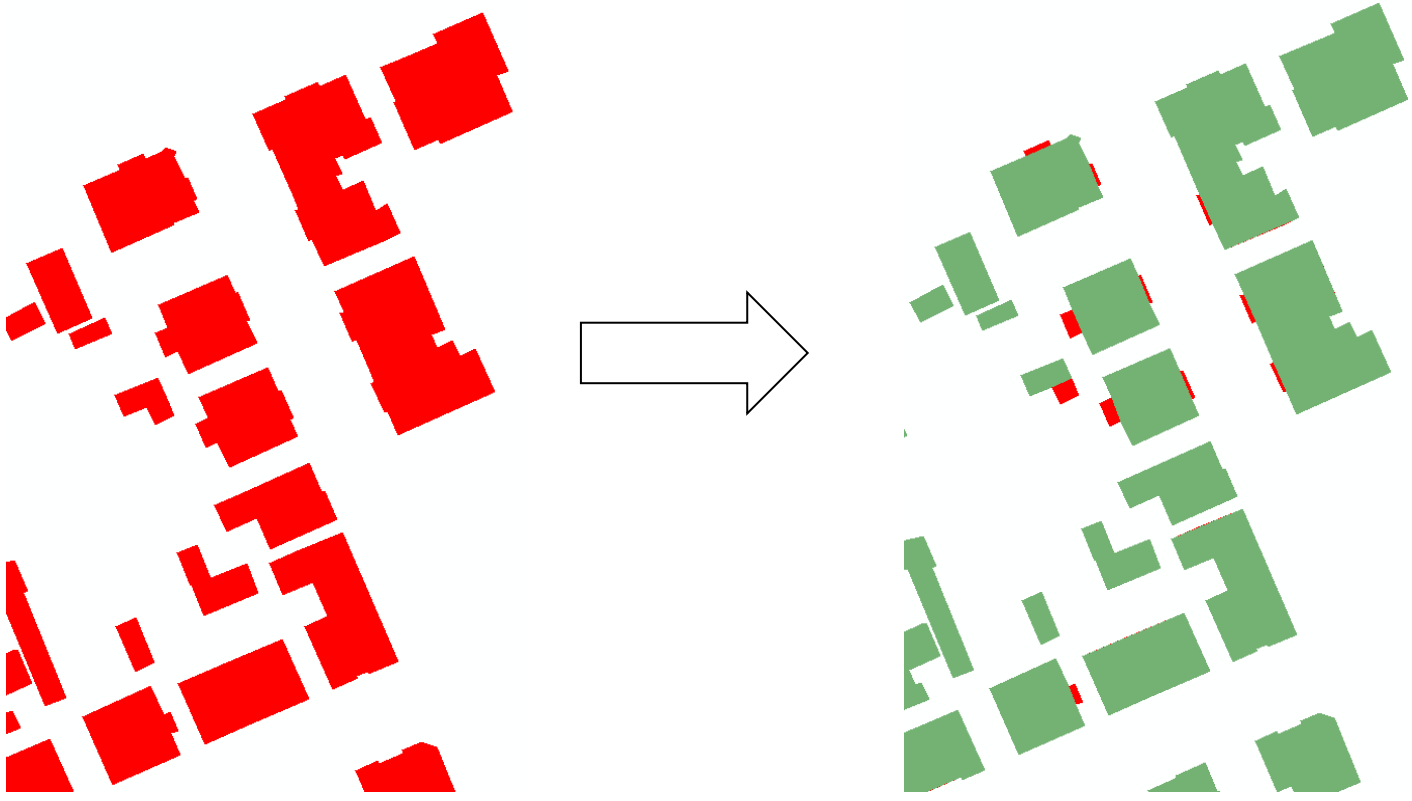
Theodor Rieche

Fakultät Geoinformation
HTW Dresden
Masterstudiengang Geoinformatik / Management

Gliederung

Motivation

gegeben: 2D Gebäudegrundrisse (Polygone)
gesucht: abgeleitete Grundrisse für gewählten Bezugs-Maßstab → Generalisierung

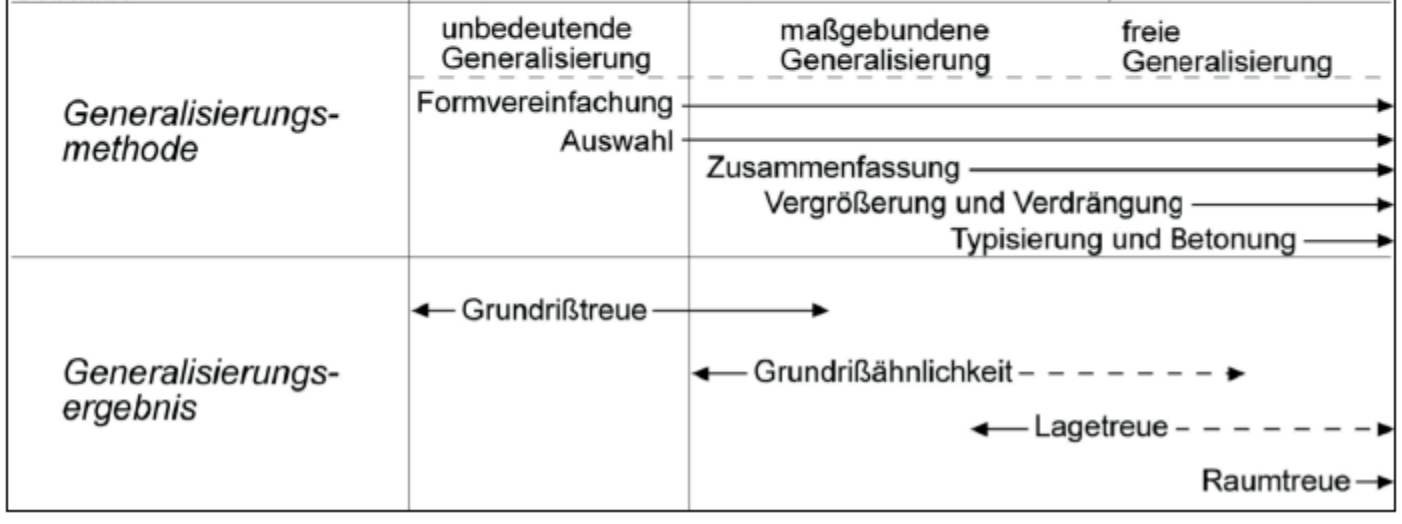


Testgebiet Gebäudepolygone von Hausumringe Deutschland (HU-DE) in Gotha

- Generalisierung von Gebäudegrundrissen
  - Grundrisstreue & Grundrissähnliche Darstellung ab 1:50.000 und größer

Gebäudedimensionen und Generalisierungskonsequenzen

Objekt	*Naturdim. in m*	*Kartendimension in mm im Maßstab* 1:5000	1:10000	1:25000	1:50000	1:100000	1:200000	1:500000
Gebäude-details	2	0,4	0,2	0,08	0,04	0,02	0,01	0,00
	4	0,8	0,4	0,16	0,08	0,04	0,02	0,01
Kleinbauten	6	1,2	0,6	0,24	0,12	0,06	0,03	0,01
Kleine und	8	1,6	0,8	0,32	0,16	0,08	0,04	0,02
mittlere Gebäude	10	2,0	1,0	0,40	0,20	0,10	0,05	0,02
Größere	20	4,0	2,0	0,80	0,40	0,20	0,10	0,04
Gebäude	50	10,0	5,0	2,0	1,0	0,5	0,25	0,1



Motivation & Einführung

- Manuelles Zeichnen nicht bezahlbar → automatischer Ablauf
- Datenbank + Bezugsmaßstab → abgeleitete Daten
- Grundkarte-Folgekarte-Prinzip *[1, Seite 25]*
- Charakteristische Form soll erhalten bleiben
- Eine spezielle Form der Linienglättung, aber mit Parallelität, Kollinearität und Rechtwinkligkeit *[1, Seite 26]*.

Generalisierungs-Einzelschritt	Verarbeitungs-priorität	mathematischer Ansatz
1 Eckversprung weglassen	1	Geradenschnitt
2 Eckversprünge zusammenfassen	2	Eliminieren des Zwischenpunktes
3 Eckversprünge weglassen	3	Geradenschnitt
Vorbau / Einsprung	4/5	Geradenschnitt
Seitenversprung weglassen	6	Geradenschnitt, Flächengleichheit
Vorbau / Einsprung betonen	7/8	Polares Anhängen, Geradenschnitt

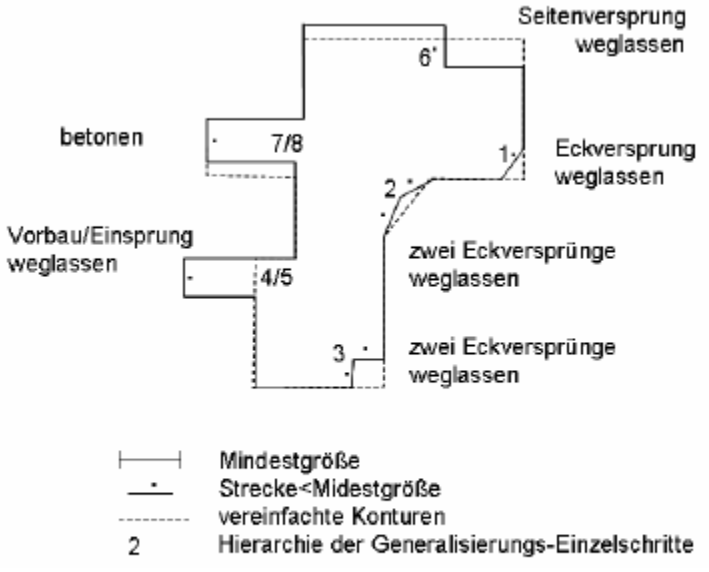


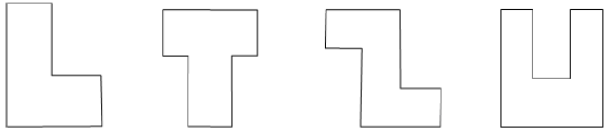
Abbildung 3-1: Generalisierungsmaßnahmen nach Staufenbiel [1973] (aus [Grünreich 1985]).

Ansätze

Ansätze

Generalisierung von Gebäudegrundrissen

- Erzwingen einer Rechtwinkligkeit / rectification
  - Siehe OpenStreetMap JOSM Editor „building generalization plug-in“, erzwingt rechte Winkel bei Winkeln zwischen 84° und 96° (Java-Implementierung) → deshalb sind OSM-Gebäude oft rechtwinklig!
- Merkmalsextraktion
  - Detektion von kleinen Vor- und Rückbauten im Grundriss
  - Prüfung hinsichtlich Minimaldimension (Fläche / Seite / Kleinseite)
  - Gegebenenfalls Eliminierung → Formvereinfachung
  - Evt. Berechnung der neuen Geometrie basierend auf Flächengleichheit
- Mustererkennung
  - Ersetzen des Grundrisses durch Basis-Formen



  - Durch Annähern an Quadrat, Rechteck, L-Form, U-Form und weitere
  - Z. Bsp. durch 3x3 Matrix (basierend auf MBR), jedes Feld anhand des Zentroid prüfen, ob innerhalb oder außerhalb des Gebäudegrundrisses
  - Oder: in 1m-Schritten die Formen an Grundriss annähern → Brute-Force Ansatz, sehr langsam…

Ansätze

- Flächenkriterium
  - Entfernen bei Unterschreitung der Minimal-Dimension
- Nachbarschafts-Analysen
  - Zusammenführen / Dissolven von adjazenten / benachbarten Gebäuden
  - Ermittlung kürzester Abstände zwischen Gebäuden zur Generierung einer gemeinsamen Siedlungsfläche / Built-Up Area (BUA)

Mögliche mathematische Manipulations-Möglichkeiten

- Über Winkel / „Windungsgeneralisierung“:
  - Innenwinkel der Stützpunkte
  - Von festem Stützpunkt Winkel zu allen anderen Stützpunkten basierend auf horizontaler Ebene, etc.
- Über Dreiecke
  - Triangulation des Polygons, …
- Über minimale Begrenzungsgeometrien wie MBR, minialer Kreis, Convex Hull, etc.
- Über Verschieben von Kanten

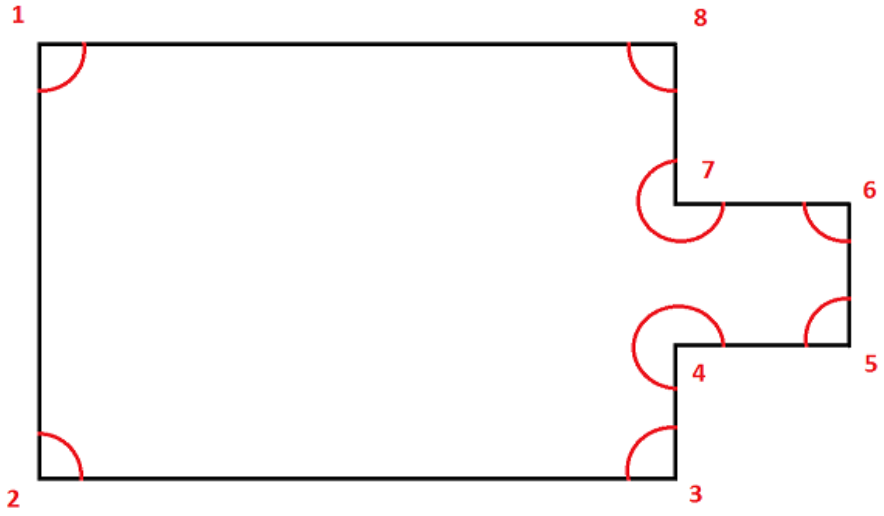
Algorithmus

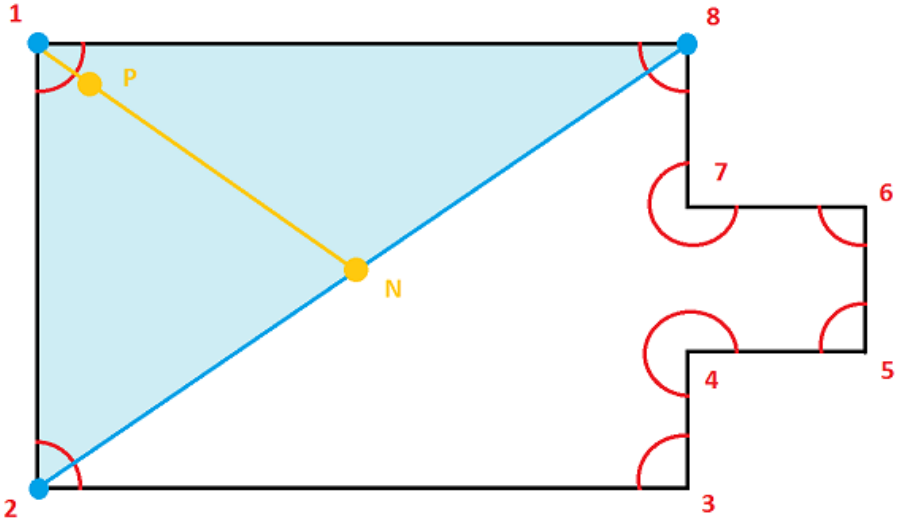
Algorithmus

- Eingabe-Polygone laden, Bezugsmaßstab als Parameter
- Minimal-Dimensionen (angelehnt an SGK No.17 [2])
  - Punktsymbol Quadratisch (schwarzer Kontur) 0,70 mm
  - Fläche Quadrat 0,35 mm x 0,35 mm = 0,1225 mm^2
  - Abhängig vom Bezugsmaßstab berechnet, könnte auch separat als Parameter implementiert werden
  - Unterschiede zwischen Bildschirm und Druck
- ID-Attribut gegebenenfalls anlegen
- Multipart to Singlepart / Explode
- Dissolve / Zusammenführen + Spatial Join (Attribute erhalten)
- Punktreduktion durch Douglas Peucker (Epsilon = 0,1 Meter)
- Selektion durch Flächenkriterium (Minimal-Dimension)
- Orientierung im Polygonzug: Ziel → gegen UZS orientieren [3]

Algorithmus

- Für jeden Stützpunkt Innenwinkel berechnen

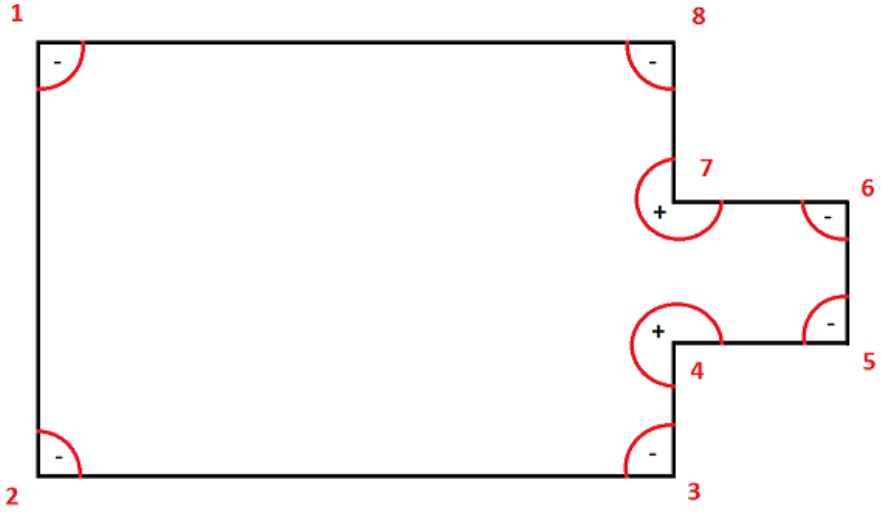




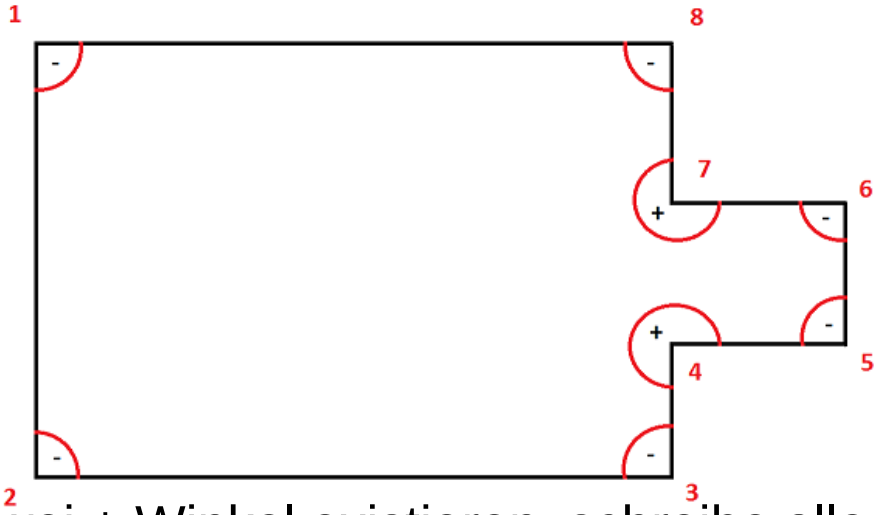
  - Über Dreieck zwischen aktuellem Punkt sowie Vorgänger und Nachfolger
  - SSS-Fall → Kosinus-Satz → Winkel berechnen
  - Ist es ein spitzer oder stumpfer Winkel? → Hilfskonstruktion
  - Prüfen, ob Dreieck außerhalb des Polygons liegt (Neupunkte N, P)
  - Falls Punkt außerhalb liegt → Winkel_neu = 2 * pi – Winkel_alt

Algorithmus

- Liste anlegen für alle Stützpunkte eine Zeile / row mit:
  - X Koordinate
  - Y Koordinate
  - Innenwinkel in Bogenmaß
  - Vorzeichen: „**+**“ bei > pi (180°), „**-**“ bei < pi
  - Boolean zunächst immer „True“, um Punkte später zu entfernen auch „False“



Algorithmus



- Wenn mind. zwei + Winkel existieren, schreibe alle Positionen / Index von + in eine neue Liste [4;7]
- Gehe jedes Intervall zwischen + Winkeln durch und prüfe, ob mindestens zwei – Winkel dazwischen liegen […; **4**;5;6;**7**;…]
- Falls ja, bilde die Fläche zwischen den begrenzenden + und den eingeschlossenen – Punkten (Merkmalsextraktion)
- Falls deren aufspannende Fläche kleiner Minimal-Dimension → Setze die eingeschlossenen Punkte auf „False“ / Löschen
- Führe die letzten 4 Schritte mit umgekehrten Vorzeichen erneut durch

Mathematische Grundlagen

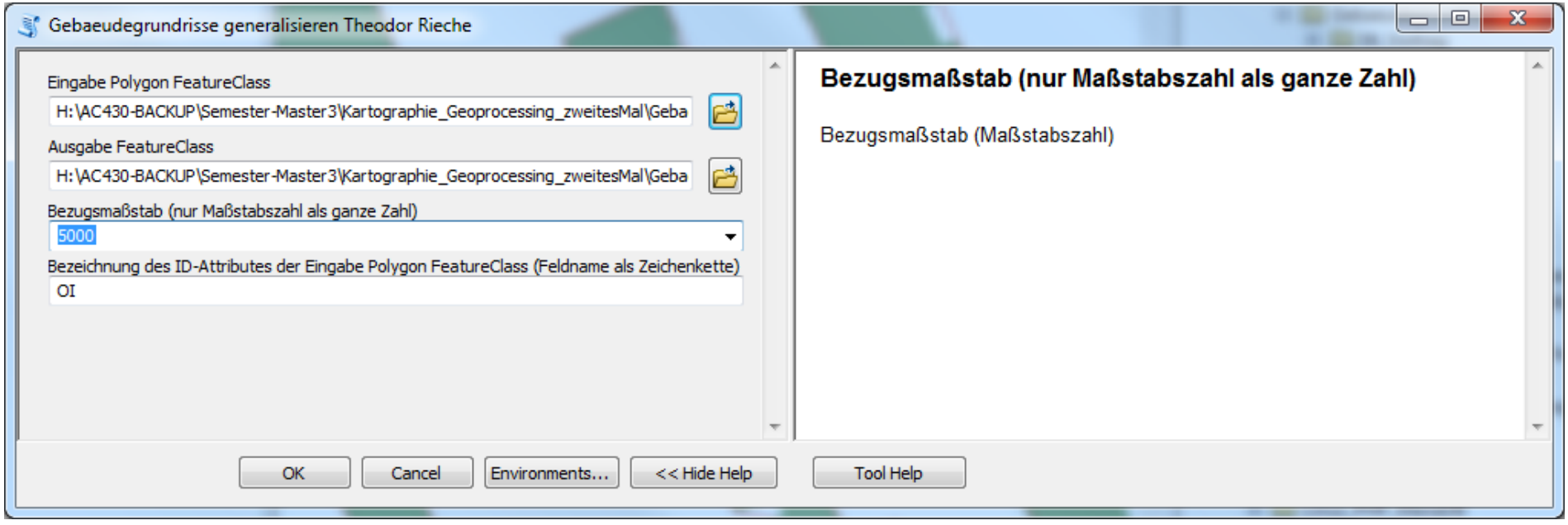
Mathematische Grundlagen

- Vektorrechnung/ Matrizen-Rechnung
- Orientierung eines Polygons/ Liste von Punkten (Quelle [1])
  - *Summe über alle Kanten: (x2-x1)(y2+y1). Wenn Ergebnis positiv → im Uhrzeigersinn. Wenn negativ → gegen UZS*
- Strecke: Satz des Pythagoras
- Kosinus-Satz im Dreieck
- Point in Polygon (ArcPy contains-Funktion)
- Douglas Peucker Algorithmus (importiert von https://stackoverrun.com/de/q/10459600 von “Momow”)
- Fläche Polygon (ArcPy area-Funktion)

Implementierung

Implementierung

- Arcpy-Toolbox mit vier Parametern
  - Eingabe Feature Class
  - Ausgabe Feature Class
  - Bezugsmaßstab
  - Feldname für ID-Feld in Eingabe Feature Class



Implementierung

- Umgesetzt in Python 2.7
- Proprietärer Ansatz unter Nutzung von ArcPy
- Software-Einsatz: PyScripter, ArcMap/ QGIS, notepad++, Excel
- 1.000 Zeilen erstellt (inkl. Kommentare und Leer-Zeilen)
- Importierte Bibliotheken: u. a. arcpy, math, copy
- Viel mit arcpy.AddMessage protokolliert
- Zwischenschritte als temp Shapefiles in Output-Ordner

Randbedingungen:

- Eingabe Daten in metrischen kartesischem CRS
- Gebäudegrundrisse als 2D-Polygone
- Aktuelle keine inneren Ringe / Löcher möglich

Herausforderungen

Herausforderungen

- Manches klappt gut, manches noch nicht
- Noch nicht praxistauglich, → experimentell

Herausforderungen

- Technologie-Wahl (OpenSource/ Geopandas+GeoJSON ODER proprietär mit ArcPy)
- Strukturierung der Thematik, Bewertung der verschiedenen Ansätze nach Sichtung in Publikationen
  - Ansätze zu 2D / 3D
  - Raster / Vektor (zB Bilderkennung in Fernerkundungsdaten)
  - „Windungsgeneralisierung“ zunächst schwer zu definieren
- Exaktes Programmieren in Python in Arrays/ Listen, Polygonen, Index…

Fazit & Ausblick

Fazit & Ausblick

- Programmier-Fertigkeiten verbessert
- Algorithmische Manipulation von Geometrien / Formen geübt
- Weites Themenfeld, viele spannende Ansätze
- Reduktion auf einen Ansatz der Windungsgeneralisierung
- Noch nicht Praxistauglichkeit erreicht

- Neue Ansätze könnten realisiert werden
- Zusätzliche Parameter im Tool
- Heuristiken / Date Science Ansätze könnten zur Formenanalyse eingesetzt werden

Quelle

[1] Kada, Martin - Zur maßstabsabhängigen Erzeugung von 3D-Stadtmodellen, Dissertation, Universität Stuttgart, 2007

[2] SGK Schweizer Gesellschaft für Kartographie, No. 17

[3] Algorithmus zur Orientierung von Polygonen: https://stackoverflow.com/questions/1165647/how-to-determine-if-a-list-of-polygon-points-are-in-clockwise-order/1165943#1165943

Alle Grafiken und Screenshots ohne Quelle sind von Theodor Rieche

Vielen Dank für Ihre Aufmerksamkeit

Quellcode Innenwinkel berechnen 1

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN
UNIVERSITY OF APPLIED SCIENCES

```
# calculate interior angles of each vertice

list_interior_angles = []
# number of '+' and '-' vertices .. will be counted later
anzahl_plus = 0
anzahl_minus = 0
for k in range(0, len(pkt_array)):
    aktuell = -1
    vorgaenger = -1
    nachfolger = -1

    if k == 0:
        aktuell = 0
        vorgaenger = len(pkt_array)-1
        nachfolger = 1

    if k == len(pkt_array)-1:
        aktuell = len(pkt_array)-1
        vorgaenger = len(pkt_array)-2
        nachfolger = 0

    if k > 0 and k < (len(pkt_array)-1) :
        aktuell = k
        vorgaenger = k - 1
        nachfolger = k + 1

    # construct triangle between i, i-1 and i+1 vertice

    # calculate angle within triangle for vertice i

    # Kosinus Satz
    seite_a = math.sqrt((pkt_array[aktuell][0]-pkt_array[vorgaenger][0])**2+(pkt_array[aktuell][1]-pkt_array[vorgaenger][1])**2)
    seite_b = math.sqrt((pkt_array[aktuell][0]-pkt_array[nachfolger][0])**2+(pkt_array[aktuell][1]-pkt_array[nachfolger][1])**2)
    seite_c = math.sqrt((pkt_array[nachfolger][0]-pkt_array[vorgaenger][0])**2+(pkt_array[nachfolger][1]-pkt_array[vorgaenger][1])**2)

    cos_gamma = (seite_a**2 + seite_b**2 - seite_c**2)/(2 * seite_a * seite_b)

    winkel_gamma = math.acos(cos_gamma)
```

Quellcode Innenwinkel berechnen 2


```
# check, if angle is interior or exterior of polygon:

# create new point N half between i-1 and i+1
N_x = (pkt_array[nachfolger][0] + pkt_array[vorgaenger][0]) / 2
N_y = (pkt_array[nachfolger][1] + pkt_array[vorgaenger][1]) / 2

# create new point P, go 1 cm from i in the direction of N
Vektor_x = N_x - pkt_array[aktuell][0]
Vektor_y = N_y - pkt_array[aktuell][1]
Vektor_betrag = math.sqrt(Vektor_x**2 + Vektor_y**2)

P_x = pkt_array[aktuell][0] + Vektor_x / Vektor_betrag * 0.01
P_y = pkt_array[aktuell][1] + Vektor_y / Vektor_betrag * 0.01

#
PointObject = arcpy.Point(P_x, P_y)

# if this new point intersects the polygon, everything is fine
angle = winkel_gamma
vorzeichen = '-'

if arcpy_array.contains(PointObject) == True:
    anzahl_minus += 1
else:
    # if not, angle will be 360°-alpha / 2pi - alpha
    angle = 2 * math.pi - winkel_gamma
    vorzeichen = '+'
    anzahl_plus += 1

del PointObject

list_interior_angles.append([pkt_array[aktuell][0], pkt_array[aktuell][1], angle, vorzeichen, 'True'])
# the last columns - after vorzeichen - shows, whether the vertice will be removed ore will be kept
```

Quellcode + Punkt 1


```
# Generalisierung  --> anhand den berechneten Innenwinkeln den Grundriss nach Merkmalen (Vorspruengen, etc.) untersuchen


# if at least two '+' angles are existing:
if anzahl_plus >= 2:

    # find position / index of '+' vertices
    list_index = []
    for k in range(0, len(list_interior_angles)):
        if list_interior_angles[k][3] == '+':
            list_index.append(k)

    # jump from '+' to '+' vertices and examine the intervalles between this points, including the '+' points before and after

    #iterate over number of '+' vertices, because it is the same number of intervalls between '+' vertices
    for i in range(0, anzahl_plus):
        # handle the first intervalls in a different way compared to the last intervall between '+' vertices
        if i < anzahl_plus - 1:
            # check, if there are minimum two '-' vertices in between
            # it is enough to calc the difference of index ... all points in between will be '-' points
            if list_index[i+1] - list_index[i] >= 3:
                # if yes: create little polygon and compare area to minimum dimension
                arr = arcpy.Array()
                arr.removeAll
                for x in range(list_index[i], list_index[i+1]+1):
                    arr.append(arcpy.Point(list_interior_angles[x][0], list_interior_angles[x][1]))

                # add first point again
                arr.append(arcpy.Point(list_interior_angles[list_index[i]][0], list_interior_angles[list_index[i]][1]))
                poly = arcpy.Polygon(arr, spatial_ref)

                # if to little, remove the points between by using "False" boolean in list
                if poly.area < akt_min_flaeche:
                    for x in range(list_index[i]+1, list_index[i+1]+1-1):
                        list_interior_angles[x][4] = 'False'
                del arr, poly
```

Quellcode + Punkt 2


```
if i == anzahl_plus - 1:
    # this is the last intervall, which reach the end of the point list.
    #so also vertices from the beginning of the list should be included


    # check, if there are minimum two '-' vertices in between
    # it is enough to calc the difference of index ... all points in between will be '-' points

    if ((len(list_interior_angles)-1) - list_index[i]) + (list_index[0]) >= 3:
        # if yes: create little polygon and compare area to minimum dimension
        arr = arcpy.Array()
        arr.removeAll
        for x in range(list_index[i], len(list_interior_angles)):
            arr.append(arcpy.Point(list_interior_angles[x][0], list_interior_angles[x][1]))
        for x in range(0, list_index[0]+1):
            arr.append(arcpy.Point(list_interior_angles[x][0], list_interior_angles[x][1]))


        # add first point again
        arr.append(arcpy.Point(list_interior_angles[list_index[i]][0], list_interior_angles[list_index[i]][1]))
        poly = arcpy.Polygon(arr, spatial_ref)

        # if to little, remove the points between by using "False" boolean in list
        if poly.area < akt_min_flaeche:
            arcpy.AddMessage('Punkte geloescht (+) - ID: ' + str(row[0]))

            for x in range(list_index[i], len(list_interior_angles)):
                list_interior_angles[x][4] = 'False'
            for x in range(0, list_index[0]+1):
                list_interior_angles[x][4] = 'False'

        del arr, poly
```

Quellcode – Punkt 1


```
# if at least two '-' angles are existing:
if anzahl_minus >= 2:

    # find position / index of '+' vertices
    list_index = []
    for k in range(0, len(list_interior_angles)):
        if list_interior_angles[k][3] == '-':
            list_index.append(k)

    # jump from '-' to '-' vertices and examine the intervalles between this points, including the '-' points before and after

    #iterate over number of '-' vertices, because it is the same number of intervalls between '-' vertices
    for i in range(0, anzahl_minus):
        # handle the first intervalls in a different way compared to the last intervall between '-' vertices
        if i < anzahl_minus - 1:
            # check, if there are minimum two '+' vertices in between
            # it is enough to calc the difference of index ... all points in between will be '+' points
            if list_index[i+1] - list_index[i] >= 3:
                # if yes: create little polygon and compare area to minimum dimension
                arr = arcpy.Array()
                arr.removeAll
                for x in range(list_index[i], list_index[i+1]+1):
                    arr.append(arcpy.Point(list_interior_angles[x][0], list_interior_angles[x][1]))

                # add first point again
                arr.append(arcpy.Point(list_interior_angles[list_index[i]][0], list_interior_angles[list_index[i]][1]))
                poly = arcpy.Polygon(arr, spatial_ref)

                # if to little, remove the points between by using "False" boolean in list
                if poly.area < akt_min_flaeche:
                    for x in range(list_index[i]+1, list_index[i+1]+1-1):
                        list_interior_angles[x][4] = 'False'
                del arr, poly
```

Quellcode – Punkt 2

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN
UNIVERSITY OF APPLIED SCIENCES

```
if i == anzahl_minus - 1:
    # this is the last intervall, which reach the end of the point list.
    #so also vertices from the beginning of the list should be included


    # check, if there are minimum two '+' vertices in between
    # it is enough to calc the difference of index ... all points in between will be '+' points

    if ((len(list_interior_angles)-1) - list_index[i]) + (list_index[0]) >= 3:
        # if yes: create little polygon and compare area to minimum dimension
        arr = arcpy.Array()
        arr.removeAll
        for x in range(list_index[i], len(list_interior_angles)):
            arr.append(arcpy.Point(list_interior_angles[x][0], list_interior_angles[x][1]))
        for x in range(0, list_index[0]+1):
            arr.append(arcpy.Point(list_interior_angles[x][0], list_interior_angles[x][1]))


        # add first point again
        arr.append(arcpy.Point(list_interior_angles[list_index[i]][0], list_interior_angles[list_index[i]][1]))
        poly = arcpy.Polygon(arr, spatial_ref)

        # if to little, remove the points between by using "False" boolean in list
        if poly.area < akt_min_flaeche:
            arcpy.AddMessage('Punkte geloescht (-) - ID: ' + str(row[0]))

            for x in range(list_index[i], len(list_interior_angles)):
                list_interior_angles[x][4] = 'False'
            for x in range(0, list_index[0]+1):
                list_interior_angles[x][4] = 'False'

        del arr, poly
```

Quellcode Update der Geometrie (nur ‚True‘)


```
        startPoint = arcpy.Point(None)
        count_true_points = 0

        # read edited points and save back to array to write finnaly back into FC
        for k in range(0, len(list_interior_angles)):
            if list_interior_angles[k][4] == 'True':

                PointObject = arcpy.Point(list_interior_angles[k][0],list_interior_angles[k][1])
                arrayPart.append(PointObject)
                startPoint = PointObject
                del PointObject
                count_true_points += 1

        # copy first point again to the end...!
        arrayPart.append(startPoint)

        arrayFeature.append(arrayPart)

    # write / update modified geometry of polygon
    row[2] = arcpy.Polygon(arrayFeature, spatial_ref)
    cursor.updateRow(row)
```