Министерство науки и высшего образования Российской Федерации

**Муромский институт (филиал)**

федерального государственного бюджетного образовательного учреждения высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(МИ ВлГУ)**

Факультет ИТР

Кафедра ИС

*ОТЧЕТ*

По НИР

тема: «Интеграция растровой и векторной карты»

Руководитель

к. т. н., доц. каф. ИС

(уч. степень, звание)

Еремеев С. В.

(оценка) (фамилия, инициалы)

(подпись) (дата)

Члены комиссии Студенты ИС-118

(группа)

Митрофанова К.Р.

(подпись) (Ф.И.О.) (фамилия, инициалы)

(подпись) (Ф.И.О.) (подпись) (дата)

Муром 2022

СОДЕРЖАНИЕ

[1 Постановка задачи 4](#_Toc98767830)

[2 Разработка алгоритма 6](#_Toc98767831)

[3 Исследовательская часть 13](#_Toc98767832)

[ЗАКЛЮЧЕНИЕ 18](#_Toc98767833)

[СПИСОК ЛИТЕРАТУРЫ 19](#_Toc98767834)

[ПРИЛОЖЕНИЕ 20](#_Toc98767835)

# 1 Постановка задачи

В данной исследовательской работе, по условию задачи, необходимо разработать алгоритм, в результате которого будет восполняться векторная карта недостающими объектами, взятыми с растрового изображения.

Исходные данные для данного алгоритма:

* векторная карта;
* изображение со спутника.



Рисунок 1 – Векторная карта



Рисунок 2 – Изображение со спутника

В качестве среды разработки использовалась географическая информационная система QGIS. Она имеет открытый исходный код и подробную документацию.

QGIS работает в Windows и в большинстве платформ Unix (включая Mac OS), поддерживает множество векторных и растровых форматов и баз данных, а также имеет богатый набор встроенных инструментов.

С помощью удобного графического интерфейса можно создавать карты и исследовать пространственные данные, а так же просматривать и накладывать друг на друга векторные и растровые данные в различных форматах и проекциях без преобразования во внутренний или общий формат.

Для реализации алгоритма использовался язык программирования PyQGIS. Данный язык позволяет запускать созданный скрипт в консоли системы QGIS, что ускорит и упростит процесс реализации алгоритма.

# 2 Разработка алгоритма

Для реализации поставленной задачи, в данной исследовательской работе, необходимо разработать алгоритм, путем которого будет создаваться векторный слой. Было разработано несколько функций:

1. Функци GetBuildings

Конечным итогом данной функции является изображение, в котором отображаются только контура зданий.

Алгоритм:

* 1. Исходное растровое изображение разделяем на 3 канала;
  2. Берем второй канал (G);
  3. Преобразуем полученный канал: все пиксели, которые больше 238 преобразуем в 0, остальные в 255;
  4. Производим медианное сглаживание изображения (cv2.medianBlur(img, 5));
  5. Наложение одного канала, которое мы получили в пункте 2, на изображение, полученное в пункте 4;
  6. Получение контуров зданий (cv2.findContours(img, cv2.RETR\_LIST, cv2.LINE\_4));
  7. Отрисовка контуров на новом изображении;
  8. К полученному изображению добавляем рамку в 20 px. Для отрисовки домов на краю.



Рисунок 3 – Контура зданий

1. Функция получения векторного слоя

Алгоритм:

* 1. Полученное изображение после предобработки разделяем на 3 канала;
  2. Берем первый канал (R);
  3. Преобразуем полученный канал: все пиксели, которые больше 238 преобразуем в 0, остальные в 255;
  4. Производим медианное сглаживание изображения

(cv2.medianBlur(img, 3));

* 1. Находим контуры зданий (cv2.findContours(img, cv2.RETR\_LIST, cv2.LINE\_4));
  2. Идем в цикле по контурам:
     1. Если контур является последним, то выходим из цикла;
     2. Создаем объект;
     3. Находим площадь контура и определяем его точки;
     4. Берем точки контура и на основании их создаем геометрию;
     5. Применяем созданную геометрию к объекту;
     6. Если контур является первым, то добавляем данную геометрию в список контуров, который необходимо отобразить и переходим к следующей итерации;
     7. Идем по списку объектов, которые хотим отрисовать;
        1. Сравниваем площадь объекта, который собираемся отобразить с площадью рассматриваемого объекта;
        2. Если площадь объекта, который мы собираемся отобразить, меньше или равна площади рассматриваемого объекта:
           1. Идем по точкам объекта, который необходимо отобразить. Если все точки этого объекта содержатся в рассматриваемом объекте, то заносим в список для удаления объект, который необходимо отобразить и выходим из цикла;
           2. Иначе возвращаемся в начало цикла;
        3. Иначе:
           1. Идем по точкам рассматриваемого объекта;
           2. Если точки этого объекта содержатся в другом объекте, то выходим из цикла;
           3. Иначе возвращаемся в начало цикла;
     8. Если рассматриваемый объект не находится внутри другого объекта, то добавляем его в список объектов, которые мы собираемся отрисовать;
  3. Удаляем из списка объектов объекты, полученные в пункте 2.6.7.2.1;
  4. Отрисовываем полученные объекты.



Рисунок 4 – Векторный слой

1. Функция интеграции растровой и векторной карты

Для интеграции растровой и векторной карты разработан алгоритм, в результате которого, отображаются недостающие объекты.

Алгоритм:

* 1. Загрузка векторной карты;
  2. Загрузка растрового изображения (cv2.imread('D:/QGIS/tiles/5\_contr.png'));
  3. Находим точки объектов растрового слоя;
  4. Сглаживаем изображение с помощью медиального сглаживания (cv2.medianBlur(img, 3));
  5. Находим контуры зданий (cv2.findContours(img, cv2.RETR\_LIST, cv2.LINE\_4));
  6. Цикл по контурам:
     1. Если контур является последним, то выходим из цикла;
     2. Создаем объект;
     3. Находим площадь контура и определяем его точки;
     4. Берем точки контура и на основании их создаем геометрию;
     5. Применяем созданную геометрию к объекту;
     6. Если контур является первым, то добавляем его геометрию в список контуров, которые необходимо отобразить и переходим к следующей итерации;
     7. Цикл по списку объектов, которые необходимо отобразить:
        1. Сравниваем площадь объекта, который мы собираемся отобразить с площадью рассматриваемого объекта;
        2. Если площадь объекта, который мы собираемся отобразить меньше или равна площади рассматриваемого объекта:
           1. Идем по точкам объекта, который необходимо отобразить. Если все точки этого объекта содержатся в рассматриваемом объекте, то заносим в список для удаления объект, который необходимо отобразить и выходим из цикла;
           2. Иначе возвращаемся в начало цикла;
        3. Иначе:
           1. Идем по точкам рассматриваемого объекта;
           2. Если точки этого объекта содержатся в другом объекте, то выходим из цикла;
           3. Иначе возвращаемся в начало цикла;
     8. Если рассматриваемый объект не находится внутри другого объекта, то добавляем его в список объектов, которые мы собираемся отрисовать;
  7. Удаляем из списка объектов объекты, полученные в пункте 3.6.7.2.1;
  8. Сравниваем растровые и векторные объекты;
  9. Цикл по векторным объектам:
     1. Цикл по растровым объектам:
        1. Цикл по точкам векторного слоя
           1. Цикл по точкам растрового объекта
           2. Сравниваем точки векторного объекта и точки растрового объекта с максимальным отклонением 10;
           3. Если они совпали, сохраняем точки в отдельном списке и удаляем их, для того, что бы ни сравнивать их в дальнейшем;
           4. Если количество совпавших точек равно количеству точек векторного объекта, то этот объект заносится в список объектов, которые не нужно отрисовывать;
           5. Удаляем объект в растровом слое, совпавшего с векторным, для того, чтобы не сравнивать его с остальными объектами;
        2. Иначе:
        3. Возвращаем удаленные точки;
  10. Отображаем объекты, индексы которые не содержатся в пункте 3.9.1.1.4.



Рисунок 5 – Векторная карта с недостающими объектами



Рисунок 6 – Итоговый результат

# 3 Исследовательская часть

В ходе научно-исследовательской работы произведено тестирование данного алгоритма.

Тест 1:



Рисунок 7 – Изображение со спутника

Векторные объекты данного теста отображаются с помощью функции получения векторного слоя. При наложении векторной карты на изображение видны недостающие объекты, которые необходимо воссоздать.



Рисунок 8 – Растровая и векторная карта

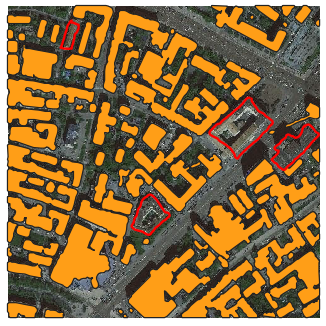


Рисунок 9 – Недостающие объекты

В результате работы данного алгоритма происходит поиск недостающих объектов и отображение их на новом слое, для более удобного визуального восприятия.



Рисунок 10 – Итоговый результат

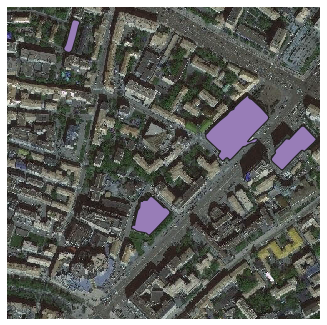


Рисунок 11 – Заполненные недостающие объекты

В ходе данного тестирования никаких ошибок не выявлено, недостающие объекты отображены на новом слое (рис. 11) .

Тест 2:

Для данного тестирования векторные объекты были отображены вручную по спутниковому снимку (рис. 7).



Рисунок 12 – Растровая и векторная карта

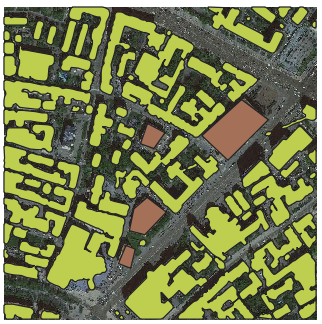


Рисунок 13 – Итоговый результат

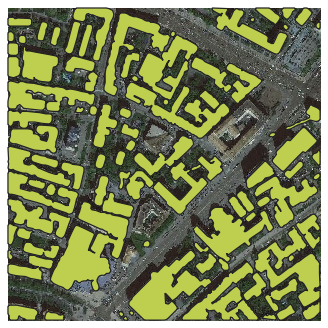


Рисунок 14 – Заполненные недостающие объекты

При сравнение векторной и растровой карты, можно использовать векторный слой нарисованный в ручную (Тест 2), тогда точки координат будут отличаться от объектов, отображенные программным способом.

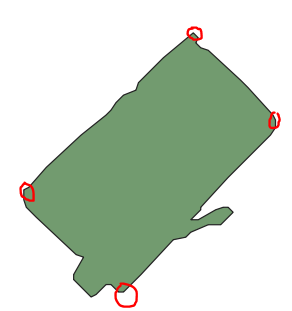
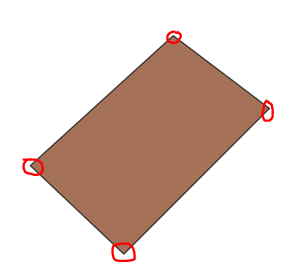


Рисунок 15 – Точки объекта

ЗАКЛЮЧЕНИЕ

В ходе выполнения научно-исследовательской работы реализован алгоритм интеграции растровой и векторной карты. По результатам тестирования данный алгоритм удовлетворяет всем поставленным условиям и работает без нареканий.

СПИСОК ЛИТЕРАТУРЫ

1. Основы геоинформатики: В 2 кн. Кн. 1: Учеб. пособие для 0-75 студ. вузов / Е.Г.Капралов, А.В.Кошкарев, В.С.Тикунов и др.; Под ред. В.С.Тикунова. — М.: Издательский центр «Академия», 2004. — 352 е., [16] с.
2. Документация QGIS// URL: https://qgis.org/ru/docs/index.html (дата обращения: 10.02.2022)
3. Документация PyQGIS// URL: https://qgis.org/pyqgis/ (дата обращения: 30.01.2022)

ПРИЛОЖЕНИЕ

Предобработка изображения

import staticmaps

import cv2

import numpy as np

import os

import random as rng

img = cv2.imread('D:/QGIS/tiles/5\_set.png')

sourceColor = np.array(img)[:-15,:]

layers = cv2.split(sourceColor)

imgnp = layers[1]

\_,imgnp = cv2.threshold(imgnp, 238, 255, cv2.THRESH\_BINARY\_INV)

imgnp\_med = cv2.medianBlur(imgnp, 5)

imgnp = cv2.bitwise\_and(imgnp\_med, imgnp)

contours, \_ = cv2.findContours(imgnp, cv2.RETR\_LIST, cv2.LINE\_4)

drawing = np.zeros\_like(imgnp)

for cnt in contours:

     M = cv2.moments(cnt)

     if M['m00'] == 0:

         continue

     if cv2.contourArea(cnt) < cv2.arcLength(cnt,True):

          continue

     cv2.drawContours(drawing, [cnt], 0, (255), cv2.FILLED)

drawing = cv2.bitwise\_and(imgnp, drawing)

img\_new = cv2.copyMakeBorder(drawing, 20, 20, 20, 20, cv2.BORDER\_CONSTANT)

cv2.imwrite(os.path.join(dir, str(k) + '\_contr.png'), img\_new)

Получение векторного слоя

import cv2

import numpy as np

img = cv2.imread('D:/QGIS/tiles/5\_contr.png')

#разделяем изображение на 3 канала

layers = cv2.split(img)

imgnp = layers[0]

#преобразуем все пиксели изображения яркость которых выше 238 в 0, остальные в значение яркости 255

\_,imgnp = cv2.threshold(imgnp, 238, 255, cv2.THRESH\_BINARY\_INV)

imgnp\_med = cv2.medianBlur(imgnp, 3)

#imgnp = cv2.bitwise\_and(imgnp\_med, imgnp)

contours, \_ = cv2.findContours(imgnp\_med, cv2.RETR\_LIST, cv2.LINE\_4)

listGeom = []

i = 0

j = 0

suri = "MultiPolygon?crs=epsg:20008&index=yes"

vl = QgsVectorLayer(suri, "treangle0", "memory")

removeList = []

for count in contours:

if (i < (len(contours) - 1)):

i+=1

fet = QgsFeature()

epsilon = 0.0001 \* cv2.arcLength(count, True)

approximations = cv2.approxPolyDP(count, epsilon, True)

a = []

b = []

for point in approximations:

b.append(QgsPointXY(point[0][0], -point[0][1]))

b.append(QgsPointXY(approximations[0][0][0],-approximations[0][0][1]))

a.append(b)

fet.setGeometry(QgsGeometry.fromPolygonXY(a))

if ( j == 0):

j+=1

listGeom.append(fet.geometry())

else:

checkGeom = 0

for geomF in listGeom:

if not geomF.equals(fet.geometry()):

if (geomF.area() <= fet.geometry().area()):

countPoints = 0

for point in geomF.asPolygon()[0]:

if fet.geometry().contains(point):

countPoints += 1

if countPoints == len(geomF.asPolygon()[0]):

checkGeom = 1

removeList.append(geomF)

continue

else:

checkGeom = 2

continue

else:

countPoints = 0

for point in fet.geometry().asPolygon()[0]:

if geomF.contains(point):

countPoints += 1

if countPoints == len(fet.geometry().asPolygon()[0]):

checkGeom = 3

break

else:

checkGeom = 2

continue

if checkGeom == 1:

listGeom.append(fet.geometry())

elif checkGeom == 2:

listGeom.append(fet.geometry())

elif checkGeom == 3:

continue

for geom in removeList:

if geom in listGeom:

listGeom.remove(geom)

i = 1

suri = "MultiPolygon?crs=epsg:20008&index=yes"

vl = QgsVectorLayer(suri, "treangle" + str(i), "memory")

for geomF in listGeom:

i += 1

fet = QgsFeature()

pr = vl.dataProvider()

vl.updateExtents()

fet.setGeometry(geomF)

pr.addFeatures([fet])

vl.updateExtents()

if not vl.isValid():

print("Layer failed to load!")

else:

QgsProject.instance().addMapLayer(vl)

Отрисовка недостающих объектов

import cv2

import numpy as np

#Возвращает одноэлементный экземпляр QgsProject

project = QgsProject.instance()

#Получить список соответствующих зарегистрированных слоев по имени слоя.

layer = project.mapLayersByName("test5")[0]

print(layer)

feat = layer.getFeatures()

#загрузка фото

img = cv2.imread('D:/QGIS/tiles/5\_contr.png')

#COLOR\_BGR2GRAY преобразование между RGB / BGR и оттенками серого, преобразование цветов

imagegray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

sourceColor = np.array(imagegray)[:-15,:]

layers = cv2.split(sourceColor)

imgnp = layers[0]

#Применяет адаптивный порог к массиву.

#cv2.THRESH\_BINARY\_INV Функция преобразует изображение в оттенках серого в двоичное изображение

\_,imgnp = cv2.threshold(imgnp, 238, 255, cv2.THRESH\_BINARY\_INV)

#медианное значение

imgnp\_med = cv2.medianBlur(imgnp, 3)

#Находит контуры в двоичном изображении.

#Функция извлекает контуры из двоичного изображения

contours, \_ = cv2.findContours(imgnp\_med, cv2.RETR\_LIST, cv2.LINE\_4)

#точки растра

#Создание списка геометрии, списка удаленных объектов и точек растра

listGeom = []

removeList = []

pointRastr = []

i=0

j=0

for count in contours:

if (i < (len(contours) - 1)):

i+=1

# Создание объект

fet = QgsFeature()

epsilon = 0.0001 \* cv2.arcLength(count, True)

approximations = cv2.approxPolyDP(count, epsilon, True)

a = []

b = []

for point in approximations:

b.append(QgsPointXY(point[0][0], -point[0][1]))

b.append(QgsPointXY(approximations[0][0][0],-approximations[0][0][1]))

a.append(b)

fet.setGeometry(QgsGeometry.fromPolygonXY(a))

if ( j == 0):

j+=1

listGeom.append(fet.geometry())

pointRastr.append(b)

else:

checkGeom = 0

for geomF in listGeom:

if not geomF.equals(fet.geometry()):

if (geomF.area() <= fet.geometry().area()):

countPoints = 0

for point in geomF.asPolygon()[0]:

if fet.geometry().contains(point):

countPoints += 1

if countPoints == len(geomF.asPolygon()[0]):

checkGeom = 1

removeList.append(geomF)

continue

else:

checkGeom = 2

continue

else:

countPoints = 0

for point in fet.geometry().asPolygon()[0]:

if geomF.contains(point):

countPoints += 1

if countPoints == len(fet.geometry().asPolygon()[0]):

checkGeom = 3

break

else:

checkGeom = 2

continue

if checkGeom == 1:

listGeom.append(fet.geometry())

pointRastr.append(b)

elif checkGeom == 2:

listGeom.append(fet.geometry())

pointRastr.append(b)

elif checkGeom == 3:

continue

# Удаление индексов

delite = []

for geom in removeList:

if geom in listGeom:

index = listGeom.index(geom)

delite.append(pointRastr[index])

#Удаление объектов

for delitePoint in delite:

if delitePoint in pointRastr:

pointRastr.remove(delitePoint)

#сравнение

i = 0

j = 0

indexList = []

tempRastRast = list(pointRastr)

#print(rastRast[31])

#print(rastVec[0])

print(pointRastr[0])

for feature in feat:

i += 1

j = 0

for points in tempRastRast:

j += 1

count = 0

tempRastRastObj = list(points)

savePoint = []

for rastVecObjPoint in feature.geometry().asPolygon()[0][0]:

for rastRastObjPoint in tempRastRastObj:

if (abs(rastVecObjPoint[0] - rastRastObjPoint[0]) >= 0 and abs(rastVecObjPoint[0] - rastRastObjPoint[0]) < 10) and (abs(rastVecObjPoint[1] - rastRastObjPoint[1]) >= 0 and abs(rastVecObjPoint[1] - rastRastObjPoint[1]) < 20):

count += 1

print(tempRastRast.index(points))

tempRastRastObj.remove(rastRastObjPoint)

savePoint.append(rastRastObjPoint)

break

print(len(rastVecObj))

if(count >= (len(feature.geometry().asMultiPolygon()[0][0]))):

print(count)

indexList.append(pointRastr.index(points))

tempRastRast.remove(points)

break

else:

for point in savePoint:

tempRastRastObj.append(point)

# Отрисовка недостоющих объектов

suri = "MultiPolygon?crs=epsg:20008&index=yes"

layer = QgsVectorLayer(suri, "treangle", "memory")

i = 0

print(indexList)

print(len(pointRastr))

#layer = project.mapLayersByName("treangle1")[0]

for geomF in pointRastr:

if i not in indexList:

a = []

fet = QgsFeature()

a.append(geomF)

fet.setGeometry(QgsGeometry.fromPolygonXY(a))

pr = layer.dataProvider()

layer.updateExtents()

pr.addFeatures([fet])

layer.updateExtents()

i += 1

QgsProject.instance().addMapLayer(layer)