mundialis GmbH & Co. KG

Sitz u. Registergericht: Bonn
Amtsgericht Bonn HRA 8528

Steuer-Nr.: 205/5823/1574
Ust.-ID-Nr.: DE300200756

Komplementärin:
mundialis Verwaltungsgesellschaft mbH

vertreten durch:
M. Eichhorn

Dokumentation

vom 06.10.2023

Entwicklung eines Verfahrens zur automatischen Detektion von Baumstandorten

Extraktion von Einzelbäumen auf Basis von True Orthophotos (TOPs) und Punktwolken sowie die Ableitung von Baumparametern und eine Veränderungsdetektion von Einzelbäumen

1.0.4

erarbeitet für:

Regionalverband Ruhr
Referat Geoinformation und Raumbeobachtung
Team Geodaten, Stadtplanwerk, Luftbilder

Kronprinzenstraße 35
45128 Essen

E-Mail: geodaten@rvr.ruhr

Dieses Projekt wird von der Bezirksregierung Münster aus Mitteln des Ministeriums für Umwelt, Naturschutz und Verkehr des Landes NRW gefördert.

erarbeitet von:

mundialis GmbH & Co. KG
Kölnstraße 99
53111 Bonn

Telefon: +49 228 - 387 580 80
Telefax: +49 228 - 962 899 57

E-Mail: info@mundialis.de
Internet: www.mundialis.de

Ansprechpersonen:
M. Metz metz@mundialis.de
M. Eichhorn eichhorn@mundialis.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Änderungen:

Tabelle 1: Änderungshistorie des Dokumentes

Datum	Autor:in	Beschreibung
17.02.2023	mundialis	Version v1.0.0
23.02.2023	mundialis	Version v1.0.1
13.07.2023	mundialis	Version v1.0.2
19.07.2023	mundialis	Version v1.0.3
06.10.2023	RVR	Version v1.0.4

Diese Dokumentation erläutert die notwendigen Arbeitsschritte zur Extraktion von Einzelbäumen auf Basis von True Orthophotos (TOPs), Punktwolken und eines digitalen Geländemodells (DGM) sowie die Ableitung diverser Baumparameter dieser Einzelbäume und die Anwendung einer Veränderungsdetektion zwischen zwei Einzelbaumlayern verschiedener Zeitpunkte. Außerdem enthält sie ein Accuracy Assessment für Validierungsdaten aus Gelsenkirchen.

1 Vorbereitung

1.1 Testumgebung

Die Software zur vorliegenden Dokumentation wurde auf einem Linux System mit 16 GB RAM, 16 CPUs und dem GRASS 8.3 Alpine Docker Image getestet.

1.2 Skripte und GRASS-Addons herunterladen

Die notwendigen Skripte und GRASS-Addons können aus dem folgenden GitHub Repository heruntergeladen werden: https://github.com/mundialis/rvr_interface.

Durch Klick auf den grünen „Code“-Button oben rechts kann das Repository entweder für die Nutzung mit der Git-Software geklont werden:

```
# Im Zielverzeichnis ausführen:
$ git clone https://github.com/mundialis/rvr_interface.git

# bzw. wenn ein SSH-Key auf GitHub hinterlegt wurde:
$ git clone git@github.com:mundialis/rvr_interface.git
```

oder als ZIP-Datei heruntergeladen und auf dem Rechner entpackt werden. Das Klonen des Repositorys hat den Vorteil, dass es einfach aktualisiert werden kann, während beim Weg über die ZIP-Datei immer das ganze Repository neu herunterzuladen ist.

Die Versionen 1.0.0 bis 3.0.0 beinhalten Skripte und GRASS-Addons zur Gebäudeextraktion und Dachbegrünung.

Ab der Version 4.0.0 sind zusätzlich GRASS-Addons zur Einzelbaumerkennung enthalten.

1.3 Benötigte Software

Die Einzelbaumerkennung sowie die Ermittlung der Baumparameter und die Veränderungsdetektion erfolgen größtenteils in **GRASS GIS**.

GRASS GIS, oder kurz GRASS (*Geographic Resources Analysis Support System*) ist ein kostenloses Open-Source Geographisches Informationssystem (GIS). Mit seinem großen Funktionsumfang wird es für die Verwaltung und Analyse von Geodaten, die Bildverarbeitung, die Erstellung von Grafiken und Karten, die räumliche Modellierung und Visualisierung verwendet. GRASS wird derzeit in akademischen und kommerziellen Einrichtungen auf der ganzen Welt sowie von vielen Regierungsbehörden und Umweltberatungsunternehmen eingesetzt.

1.3.1 GRASS GIS-Basics

Zur Nutzung von GRASS GIS sind einige grundlegende Kenntnisse über die Software nötig.

GRASS GIS unterscheidet sich von anderen GIS-Anwendungen wie ESRI ArcGIS oder QGIS, in denen die Nutzenden sofort verschiedene Daten aus verschiedenen Datenquellen in verschiedenen Projektionen laden und mit der Arbeit an einem Projekt beginnen können. Beim Starten von GRASS GIS müssen die Nutzenden zunächst das Arbeitsprojekt definieren, in dem die GRASS-Sitzung arbeiten soll.

Beim Start von GRASS GIS sind drei Voraussetzungen erforderlich (über den Startbildschirm oder die Kommandozeile einrichtbar):

- **Datenbankverzeichnis:** Ein Verzeichnis auf der lokalen oder Netzwerkplatte, das alle Daten enthält, auf die GRASS GIS zugreift. Dies ist üblicherweise ein Verzeichnis namens „*grassdata*“, das sich im individuellen Home-Verzeichnis befindet.
- ***Location*:** Spielt die Rolle eines "Projekts". Alle Geodaten, die innerhalb einer *Location* gespeichert sind, müssen das gleiche räumliche Koordinatensystem haben (GRASS GIS unterstützt aus verschiedenen Gründen keine *on-the-fly*-Projektion, kann aber beim Import reprojizieren).
- ***Mapset*:** Enthält aufgabenbezogene Daten innerhalb eines Projekts. Dies unterstützt die Organisation der Daten in logischen Gruppen oder ermöglicht die parallele Arbeit mehrerer Nutzenden am selben Projekt.

Ein weiterer Unterschied zu anderen GIS-Anwendungen ist das Konzept der ***Computational Region*.** Diese legt für Berechnungen und Analysen von Rasterdaten den räumlichen Bereich fest, für den die Berechnung durchgeführt wird. Das heißt, mit der *Region* wird die Ausdehnung und Auflösung des Ergebnisrasters bestimmt. Die aktive *Region* kann mit dem GRASS-Befehl *g.region* festgelegt werden.

Zum weiteren Verständnis der GRASS GIS-Basics kann zum Beispiel dieses Intro oder diese GRASS Manual-Seite konsultiert werden.

Weitere hilfreiche Links zum Umgang mit GRASS sind:

- GRASS GIS Homepage
- GRASS GIS 8.3 Reference Manual
- GRASS User Wiki Installation Guide
- GRASS GIS Wiki
- GRASS GIS Tutorial

1.3.2 Einrichten und Starten von GRASS in Docker und Linux

Im ersten Schritt muss zunächst GRASS GIS eingerichtet werden. Alle Daten, die in GRASS GIS verwendet werden, liegen in einer Datenbank (*grassdb*) in GRASS-spezifischen Formaten. Diese Datenbank kann als normaler Ordner neu erstellt werden. Im gewünschten Verzeichnis, in dem die GRASS-Datenbank angelegt werden soll (Achtung: diese kann im Laufe der Analyse sehr groß werden!), muss ein neuer Ordner erstellt werden (üblicherweise als *„grassdata“* im Home-Verzeichnis):

```
$ mkdir grassdata
```

Die Erstellung der Datenbank ist nur einmalig zu Beginn notwendig. Besteht bereits eine *grassdb*, kann auch diese verwendet werden.

Zur Nutzung der Baummodule wird GRASS GIS in Docker verwendet. Hierfür muss Docker auf dem System installiert sein. Genutzt wird das offizielle und öffentlich zur Verfügung stehende Docker Image von GRASS 8.3. Es enthält alle für die Baummodule notwendigen Abhängigkeiten wie Python3, GDAL und PDAL. Zunächst muss aber ein lokales Docker Image gebaut werden. Dies geschieht im Wurzelverzeichnis des RVR-Interfaces, in dem auch ein Dockerfile liegt. Dieser Schritt muss nur einmal ausgeführt werden:

```
# Navigieren zum Wurzelverzeichnis des RVR-Interfaces:
$ cd /pfad/zum/interface

# Aufbauen des lokalen Docker Images
# Der Punkt am Ende bedeutet, dass Docker nach dem Dockerfile im aktuellen
# Verzeichnis suchen soll:
$ docker build -t rvr_interface:latest .
```

Statt *latest* kann auch das Datum zum Zeitpunkt der Erstellung (*231004* o. ä.) angegeben werden.

Auf Basis dieses Images kann ein Docker-Container gestartet werden. Die Verzeichnisse, hier die GRASS-Datenbank und die Input-Daten, werden dabei mit -v in den Container gemountet:

```
# Starten des Docker-Containers
$ docker run -it -v /pfad/zu/grassdata:/ -v /pfad/zu/Eingangsdaten:/mnt/data
rvr_interface:latest sh

# Oder alternativ
$ docker run -it -v /pfad/zu/grassdata:/ -v /pfad/zu/Eingangsdaten:/mnt/data
rvr_interface:231004 sh
```

Es ist nicht notwendig, das Docker Image als root (sudo docker run …) zu starten. Alternativ können die Nutzenden der Gruppe „docker“ hinzugefügt werden:

```
# Nutzende der Gruppe „docker“ hinzufügen
$ sudo usermod -a -G docker <username>

# überprüfen mit
$ id
```

Solange GRASS GIS innerhalb des Docker-Containers genutzt wird, werden die Pfade im Docker anstelle der lokalen Pfade verwendet. Nach dem obigen Einmounten sind das die folgenden Stammpfade:

- Pfad zur grassdb: */grassdb/…*
- Pfad zu den Input-Daten: */mnt/data/…*

Diese müssen also beim Durchgehen der Schritte aus der Dokumentation ggf. angepasst werden.

Zunächst wird eine neue ***Location*** mit dem EPSG-Code 25832 (Projektion: ETRS89 / UTM zone 32N) angelegt. Alle darin importierten Datensätze verfügen dann über dieselbe Projektion:

```
# Allgemeines Schema zum Erstellen einer neuen GRASS GIS Location
$ grass -c epsg:epsg-code /grassdb/neuer_location_name

# Erstellen einer neuen GRASS GIS Location mit dem Namen „location_25832“
$ grass -c epsg:25832 /grassdb/location_25832
```

Im selben Terminal wird nun GRASS GIS gestartet, hier erfolgen die weiteren Schritte. Auch in einer geöffneten GRASS-Sitzung können weiterhin alle Terminal-Befehle ausgeführt werden (z. B. Verzeichnisse wechseln, *bash*-Skripte starten, etc.).

Innerhalb der GRASS-*Locations* gibt es ***Mapsets***, die die Sammlung von thematisch zusammenhängenden Datensätzen ermöglichen. Bei der Erstellung einer neuen *Location* wird automatisch das PERMANENT-***Mapset*** erstellt. Es ist jedoch sinnvoll, ein selbst benanntes **neues *Mapset*** zu erstellen:

```
# Neues Mapset namens “gelsenkirchen_2020” erstellen und dort hinein wechseln
$ g.mapset -c gelsenkirchen_2020

# TIPP: Für alle GRASS-Befehle können Sie <Befehl> --help ausführen, um
# mehr über die Benutzung des Befehls zu erfahren, z.B.:
$ g.mapset --help
```

Alternativ kann auch direkt beim Starten von GRASS GIS ein neues *Mapset* erstellt werden:

```
# Allgemeines Schema zum Erstellen eines neuen Mapsets in einer bestehenden
# Location
$ grass -c /grassdb/location_name/neuer_mapset_name

# Erstellen eines neuen Mapsets „gelsenkirchen_2020“ in der Location „loca-
tion_25832“
$ grass -c /grassdb/location_25832/gelsenkirchen_2020
```

Nun befinden sich die Nutzenden in GRASS GIS innerhalb der (neuen) *Location* (im Beispiel: “location_25832”) und des neu angelegten *Mapsets* (im Beispiel: „gelsenkirchen_2020“). Von hier aus können die im weiteren Verlauf in GRASS GIS stattfindenden Schritte ausgeführt werden.

Der folgende Befehl öffnet GRASS GIS zu einem anderen Zeitpunkt in einer bestimmten *Location* und *Mapset* erneut:

```
# Allgemeines Schema zum Öffnen von GRASS in bestimmten Mapsets
$ grass /pfad/zu/grassdata/yourlocation/yourmapset

# Wiederaufnahme einer GRASS-Sitzung in der Location “location_25832” und dem
# Mapset “gelsenkirchen_2020”
$ grass /grassdb/location_25832/gelsenkirchen_2020
```

Zum Umbenennen oder Löschen eines bestehenden *Mapsets* kann direkt der Ordner des *Mapsets* in der *grassdb* umbenannt bzw. gelöscht werden, z. B.:

```
# Allgemeines Schema zum Umbenennen/Löschen von GRASS Mapsets
$ mv /pfad/zu/grassdata/yourlocation/yourmapset /pfad/zu/grassdata/yourloca-
tion/new_name
$ rm -rf /pfad/zu/grassdata/yourlocation/yourmapset
```

1.3.3 Ändern der Eigentumsverhältnisse der erstellten GRASS-Daten

Der GRASS GIS Docker-Container läuft mit Root-Rechten. Daher werden die im Docker erstellten Daten als root erstellt, weshalb das Zugreifen auf diese Daten nicht wie gewohnt von allen Nutzenden möglich ist. Um z. B. eine im Docker erstellte GRASS GIS *Location/Mapset* von außerhalb des Dockers öffnen zu können, müssen vorher ggf. die Nutzenden und die Gruppe außerhalb des Dockers angepasst werden. Dies kann z. B. über die folgenden Befehle erfolgen:

```
# Abfrage der Rechte auf die gesamte GRASS-Datenbank
$ ls -lah /pfad/zu/grassdata
drwxr-xr-x 14 root root   4,0K Jan 31 13:43 location_erstellt_im_docker
drwxrwxr-x  5 USER GRUPPE 4,0K Jan 24 15:36 location_erstellt_außerhalb_docker

# Anpassen der Nutzenden und der Gruppe (beide können aus der Antwort zu
# dem obigen ‚ls‘ an der anderen Location entnommen werden und
# dementsprechend gesetzt werden)
$ sudo chown USER:GRUPPE -R /pfad/zu/grassdata/location_erstellt_im_docker
```

1.4 Benötigte Datensätze

Für die Einzelbaumextraktion werden die folgenden Datensätze benötigt:

- Gebiet:
  - Es wird ein Gebiet im Vektordatenformat (z. B. Shapefile, Geopackage) benötigt, für das die Prozessierung ausgeführt werden soll.
  - Alle anderen Datensätze müssen dieses Gebiet vollständig abdecken.
- TOPs:
  - Die benötigten True Orthophotos (TOPs) im .tif Format mit vier Farbkanälen (RGBI) müssen innerhalb eines gemeinsamen Verzeichnisses vorliegen (in diesem Verzeichnis dürfen keine weiteren Dateien enthalten sein).

- Punktwolken:
  - Die benötigten Punktwolken im .laz Format müssen in einem gemeinsamen Ordner vorliegen (in diesem Verzeichnis dürfen keine weiteren Dateien enthalten sein).
- ggf. Digitales Geländemodell (DGM; engl. digital terrain model (DTM)):
  - Falls nicht das NRW-DGM1 zur nDOM-Erstellung genutzt werden soll, kann auch ein individuell definiertes DGM mit einer idealen Auflösung von 0,5 - 1,0 m verwendet werden. Dieses muss in einem GDAL-konformen Format vorliegen (z. B. GeoTIFF, ASCII XYZ).

Für die Ableitung diverser Baumparameter werden folgende Daten benötigt:

- Hausumringe (Vektor-Polygone):
  - Zur Berechnung des Abstandes von Einzelbaumkronen zum nächsten Gebäude werden entsprechende Daten benötigt, die die Hausumringe enthalten. Diese können in einem beliebigen Vektordatenformat (z. B. Shapefile, Geopackage) vorliegen. Die Daten können selbst bereitgestellt (z. B. ALKIS-Gebäudedatensatz) oder die Gebäudedaten mit Hilfe des Import-Addons von OpenGeodata.NRW heruntergeladen werden. Achtung: Das Tool bezieht immer nur die aktuellsten Daten und keine historischen. Historische Daten müssen vorab selbst über OpenGeodata.NRW heruntergeladen werden.

1.4.1 Import und Aufbereitung der Datensätze in GRASS GIS

Für den Import und die Aufbereitung der Eingangsdaten wurde das GRASS-Addon *m.import.rvr* entwickelt. Dieses fasst mehrere Arbeitsschritte zum Datenimport und zur Aufbereitung zusammen, wobei auch die Möglichkeit besteht, Gebäudedaten von OpenNRW bei Bedarf automatisch herunterzuladen. Das Import-Tool kann ebenso für das Gebäude-Tool (s. Dokumentation „Gebäude- und Dachbegrünungsdetektion“) genutzt werden. Nachfolgend findet sich eine Übersicht der Aufgaben des Import-Addons:

- Importieren der benötigten Datensätze
  - Gebiet, TOPs, 2,5D-Punktwolken, DGM, ggf. Hausumringe
- Aufbereitung der TOPs
  - Importieren der einzelnen TOP-.tif-Dateien, die in dem angegebenen Gebiet liegen, und Resampling dieser auf 0,2 m Auflösung
  - Zusammenfassen der einzelnen TOP-Tiles zu einem virtuellen Raster (VRT). Jeder Kanal liegt als eigene Rasterkarte (*top_red_02*, *top_green_02*, *top_blue_02* und *top_nir_02*) im Mapset. Diese VRT-Karten setzen sich wiederum aus den einzelnen TOP-Tiles zusammen, z. B *top_red_02* aus verschiedenen anderen *top_*_02*.

- Aufbereitung der 2,5D-Punktwolken
  - Importieren der 2,5D-Punktwolken in GRASS GIS und Verarbeitung zu einem Digitalen Oberflächenmodell (DOM; engl. *digital surface model* (DSM)) im Rasterformat mit 0,5 m Auflösung.
  - Importieren der .laz-Tiles als DOM-Raster und anschließend Zusammensetzung zu einem Gesamtraster. Dies geschieht mit Hilfe des Import-Addons, das intern das Addon *r.in.pdal.worker* aufruft, um die einzelnen .laz-Dateien auch parallel importieren zu können.
- Berechnung des NDVI
  - Berechnung des Normalised Difference Vegetation Index (NDVI)-Rasters nach dem TOP-Import
  - Der NDVI benötigt den roten und den nahinfraroten Kanal des TOP-Mosaiks, die zuvor im Import-Addon importiert und resampled wurden.
  - Der Wertebereich wird auf 8-Bit Integer Werte (0 - 255) skaliert, um das Raster performant zu halten.
- Berechnung des normalisierten Digitalen Oberflächenmodells (nDOM; engl. *normalised digital surface model* (nDSM))
  - Zur Berechnung des nDOM (der Differenz aus DOM und DGM) wird das Addon *r.import.ndsm_nrw* im Import-Addon aufgerufen. Dieses interpoliert eventuelle NoData-Lücken im importierten DOM, die durch zu geringe Dichte der Punktwolken auftreten können, berechnet aus vorhandenem DOM und einem DGM das nDOM und resampled es auf die aktuelle *Region*.
  - Zur Berechnung des nDOM wird entweder ein individuell definiertes DGM oder das NRW-DGM verwendet. Zur Nutzung eines eigenen DGMs kann der Parameter *dsm_dir* im Import-Addon angegeben werden. Wird kein DGM definiert, bezieht das Addon automatisch die benötigten Tiles des NRW-DGM.
  - Das DGM für das Verbandsgebiet wird sowohl vom RVR als auch vom Land NRW gekachelt im XYZ-Format zur Verfügung gestellt. Dabei ist zu beachten, dass sich die XY-Koordinaten hier nicht wie allgemein üblich auf die Pixelmitte beziehen, sondern auf die linke untere Ecke eines Pixels. Wenn dies nicht beachtet wird, kann es zu einem Versatz um einen halben Pixel kommen, außerdem passen die Grenzen dann nicht mehr zu den Grenzen der TOP-Kacheln.
  - Bei der Nutzung eines individuell definierten DGMs wird dessen Versatz wie folgt gehandhabt:
    - Bei der Nutzung von XYZ-Tiles wird der Versatz vom Import-Addon korrigiert (Auflösung des DGMs muss angegeben werden).
    - Bei der Nutzung eines GeoTiffs wird von einer vorher erfolgten Korrektur ausgegangen und das GeoTiff „*as is*“ eingeladen.

Im Folgenden werden die Parameter des Import-Addons genauer aufgeschlüsselt, die für die Einzelbaumerkennung relevant sind:

Eingangsparameter:

- ***area***: Pfad zur Vektordatei, die das zu berechnende Gebiet enthält.
- ***reference_buildings_file***: Pfad zur Vektordatei mit Gebäuden für die Berechnung der Distanz der detektierten Bäume zum nächsten Gebäude.
- ***top_dir***: Pfad zum Ordner, in dem die TOP-GeoTiffs liegen.
- ***top_tindex***: Pfad zum / für einen Tileindex für die TOPs, die im *top_dir* liegen. Der Tileindex muss ein Attribut *location* haben, in dem der absolute Pfad zu den TOPs (innerhalb des Dockers) steht. Dieser wird entweder verwendet, sofern die Datei existiert und er nicht bei jedem Lauf des Addons erstellt werden muss, oder gespeichert, wenn die Datei nicht existiert, damit er für zukünftige Läufe wiederverwendet werden kann. Ändert sich hingegen die Auflösung der Eingangsdaten (z. B. zwischen zwei Bildflugjahren von 10 cm auf 7,5 cm), muss dieser neu berechnet werden (optional).
- ***dsm_dir***: Pfad zum Ordner, in dem die DOM-LAZ liegen.
- ***dsm_tindex*:** Pfad zu einem / für einen Tileindex für die DOM LAZ-Dateien, die im *dsm_dir* liegen (optional). Der Tileindex muss ein Attribut *location* haben, in dem der absolute Pfad zu den .laz-Daten (innerhalb des Dockers) steht. Dieser wird entweder verwendet, sofern die Datei existiert und er nicht bei jedem Lauf des Addons erstellt werden muss, oder gespeichert, wenn die Datei nicht existiert, damit er für zukünftige Läufe wiederverwendet werden kann. Das erste Erstellen des Indexes *dsm_tindex* kann lange dauern, weshalb es sinnvoll ist, vor allem diesen Index wiederzuverwenden. Ändert sich hingegen die Auflösung der Eingangsdaten (z. B. zwischen zwei Bildflugjahren von 10 cm auf 7,5 cm), muss dieser neu berechnet werden (optional).
- ***dtm_file***: Pfad zur DGM-Datei oder einem Ordner mit mehreren DGM-Kacheln im Format GeoTiff (Import und Resampling ohne Korrektur) oder XYZ (Import und Shift, um die Angabe der unteren linken Ecke statt des Pixelzentrums zu korrigieren, hierfür wird die *dtm_resolution* benötigt); auch ein Pfad zu einem Ordner mit mehreren XYZ-Dateien kann angegeben werden (optional).
- ***dtm_resolution***: Original-Auflösung des DGM, damit bei Verwendung einer DGM-XYZ-Datei oder mehreren DGM-XYZ-Dateien in einem Ordner eine Korrektur des Versatzes durchgeführt werden kann (optional; notwendig, wenn *dtm_file* als XYZ-Datei gegeben).
- ***dtm_tindex***: Pfad zum / für einen Tileindex für die DTM-XYZ-Dateien, die im *dtm_file* Ordner liegen. Der Tileindex muss ein Attribut *location* haben, in dem der absolute Pfad zu den XYZ-Daten (innerhalb des Dockers) steht. Dieser wird entweder verwendet, sofern die Datei existiert, damit er nicht bei jedem Lauf des Addons erstellt werden muss, oder gespeichert, wenn

die Datei nicht existiert, damit er für zukünftige Läufe wiederverwendet werden kann. Ändert sich hingegen die Auflösung der Eingangsdaten (z. B. zwischen zwei Bildflugjahren von 10 cm auf 7,5 cm), muss dieser neu berechnet werden (optional).

- ***type***: Analysetyp, für den die benötigten Daten importiert werden sollen: „einzelbaumerkennung“.
- ***-c***: Flag zum Prüfen, ob alle für den Analysetyp erforderlichen Input-Daten importiert werden können, ohne dabei den eigentlichen Import zu starten.
- ***-b***: Flag für den Download der Referenzgebäudedaten von OpenNRW, sofern der entsprechenden Parameter *reference_buildings_file* nicht gesetzt ist (Achtung: Es steht immer nur der aktuellste Datensatz zur Verfügung).

Ein Teil der Berechnung im Import-Modul kann parallel erfolgen. Für die Konfiguration der Parallelisierung können folgende Parameter gesetzt werden:

- ***memory***: Arbeitsspeicher in MB, der dem Addon zur Verfügung stehen soll (optional, Defaultwert ist 300 MB)
- ***nprocs:*** Anzahl der Kerne für die Parallelprozessierung (optional, Defaultwert ist -2 (Anzahl der verfügbaren Kerne minus 1), für serielle Prozessierung auf 1 setzen)

```
# m.import.rvr ausführen:
$ DFOLDER=/mnt/data
$ m.import.rvr memory=300 type=einzelbaumerkennung \
   area=${DFOLDER}/Gebiet/gelsenkirchen.gpkg\
   top_dir=${DFOLDER}/2020_Sommer/TOP/ \
   dsm_dir=${DFOLDER}/2020_Sommer/Punktwolke_2_5D_RGBI/ \
   dtm_dir=${DFOLDER}/2020_Sommer/DGM/dgm1_gelsenkirchen_2020_corrected.tif
   reference_buildings_file=${DFOLDER}/Gebaeude/gelsenkirchen_2020_hausum-
ringe.gpkg
```

Hinweis:
Dieser Schritt kann aufgrund der Größe der Eingangsdateien und gemounteten Datenverzeichnisse viele Stunden in Anspruch nehmen.
Zur Vermeidung von Fehlern empfehlen wir für den Arbeitsspeicher den Defaultwert von 300 MB. Für *memory* erfolgt dann keine Angabe. Darüber hinaus ist es sinnvoll, die maximale Anzahl der verfügbaren Kerne minus 1 anzugeben (Default für *nprocs*), um die Rechenzeit gering zu halten.

Das Import-Addon importiert dann die folgenden Daten für die Einzelbaumerkennung:

- ***study_area:*** eine Vektorkarte des Gebietes
- ***reference_buildings:*** die Hausumringe als Vektorkarte, die entweder angegeben wurden oder von OpenNRW heruntergeladen werden

- ***top_red_02, top_green_02, top_blue_02, top_nir_02:*** die Rasterkarten der einzelnen Kanäle des TOP-Mosaiks
- ***top_ndvi_02:*** die Rasterkarte des berechneten und skalierten NDVI
- ***dsm_02:*** die Rasterkarte des importierten DOMs
- ***ndsm:*** die Rasterkarte des berechneten nDOMs

2 Analyse

Das Multi-Modul *m.analyse.trees* umfasst die Addons *r.trees.peaks*, *r.trees.mltrain*, *r.trees.mlapply*, *r.trees.mlapply.worker*, *r.trees.postprocess*, *v.trees.param, v.trees.param.worker, v.trees.species, v.trees.cd* und *v.trees.cd.worker*.

Die Installation des Multi-Addons installiert automatisch die oben genannten Einzelbaumanalyse-Module sowie ihre zugehörigen Worker-Module. Die Worker-Module werden indirekt durch die Hauptmodule aufgerufen und dienen der parallelen Prozessierung über ein Gitter. Durch das sogenannte Tiling erfolgt ein Teil der Berechnung über mehrere Kacheln.

2.1 Tiling

Beim Tiling wird das Untersuchungsgebiet in mehrere Kacheln zerschnitten und der Analyseprozess Kachel für Kachel gerechnet. Dies gilt allerdings nur für einen Teil der Berechnungen. Manche Schritte, z. B das Trainieren des Machine Learning Modells, werden weiterhin für das Gesamtgebiet durchgeführt, damit sich das Modell auf das Gesamtgebiet bezieht. Die Kantenlänge des Gitters und damit der einzelnen Kacheln kann individuell gewählt werden, standardmäßig liegt sie bei 1000 m.

Zusätzlich kann der Parameter ***nprocs***, also die Anzahl der parallelen Prozesse, festgelegt werden. Hiernach richtet sich, wie viele Kacheln gleichzeitig berechnet werden. Für eine serielle Prozessierung ohne Parallelisierung sollte der *nprocs*-Parameter auf 1 gesetzt werden. Standardmäßig ist *nprocs* auf -2 gesetzt. Das bedeutet, es wird die Anzahl der verfügbaren Kerne ermittelt und von dieser Anzahl 1 abgezogen, sodass wenig rechenintensive Operationen, wie z. B. das Monitoring des Analyseprozesses, weiterhin durchgeführt werden können. Wenn mehrere Prozesse gleichzeitig auf der Maschine laufen, sollte der *nprocs*-Parameter deutlich herabgesetzt werden, sodass weniger Kerne pro Prozess genutzt werden und die Rechenlast pro Prozess geringer ist.

Gleiches gilt auch für den *memory*-Parameter. Dieser sollte beim Starten mehrerer Prozesse gleichzeitig nicht auf dem maximalen Wert der Maschine liegen, sondern herabgesetzt werden.

2.2 Einzelbaumdetektion

Unter einem einzelnen Baum wird hier die Baumkrone eines Baumes verstanden. Konkret werden Baumkronenpixel einem wahrscheinlichen Baum zugeordnet und zu einer individuellen Baumkrone zusammengefasst.

2.2.1 Einzelbaumextraktion

Die Identifikation von Einzelbäumen erfolgt in mehreren Schritten:

1. Geomorphologische Extraktion von Baumgipfeln
2. Erstellung von Trainingsdaten für das Machine Learning und das Training eines Modells
3. Baumklassifikation mit Machine Learning
4. Zusammenfassen der klassifizierten Baumpixel zu Einzelbäumen

Bevor diese Schritte ausgeführt werden, muss einmal die GRASS-Region mit *g.region rast=ndsm* gesetzt werden.

Die geomorphologische Extraktion von Baumgipfeln erfolgt mit dem Addon *r.trees.peaks*. Dabei wird jedem Baumgipfel eine eigene ID vergeben. Die Ergebnisse dieses Addons sind eine Rasterkarte mit der am nächsten gelegenen Baumgipfel-ID, eine Rasterkarte mit geomorphologischen Gipfeln und Bergrücken und eine Rasterkarte mit dem Gefälle des nDOMs.

Die Nähe der Baumgipfel-ID und entsprechende Zuweisung der Pixel erfolgt mit einer sogenannten „Watershed Segmentation“, die das Gefälle des nDOMs als Kostenfaktor benutzt. Diese Objekte beinhalten nicht nur Bäume, sondern auch andere höhere Objekte. Im Folgenden werden die **Parameter** dieses Addons genauer aufgeschlüsselt:

Eingangsparameter:

- ***ndsm***: Name des nDOM-Rasterdatensatzes
- ***forms_res***: Auflösung zur Berechnung geomorphologischer Formen, default 0,8 Meter

Ausgangsparameter:

- ***nearest***: Output Rasterkarte mit der ID des nächstgelegenen Baumgipfels
- ***peaks***: Output Rasterkarte mit geomorphologischen Gipfeln und Bergrücken
- ***slope***: Output Rasterkarte mit dem Gefälle des nDOMs

Für die Konfiguration der Parallelisierung können folgende Parameter gesetzt werden:

- ***tile_size***: Kantenlänge der Gitterkacheln für das Tiling in m (Defaultwert ist 2000 m)
- ***memory***: Arbeitsspeicher in MB, der dem Addon zur Verfügung stehen soll (optional, Defaultwert ist 300 MB)

Beispiel für den Aufruf von *r.trees.peaks*:

```
# Zuerst die Region setzen:
$ g.region rast=ndsm

# r.trees.peaks mit Standard-Einstellungen ausführen:
$ r.trees.peaks ndsm=ndsm nearest=nearest_tree peaks=tree_peaks \
   slope=ndsm_slope memory=1000
```

Die Erstellung von Trainingsdaten für das Trainieren eines Modells und das Machine Learning erfolgt mit dem Addon *r.trees.mltrain*. Dabei werden die vier Farbkanäle, mehrere abgeleitete normalisierte Differenzen der Bänder und das nDOM als Input-Variablen für ein Random-Forest-Modell (cross-validation: 5, Mindestanzahl Samples pro Baum-Blatt: 5, Anzahl Bäume: 100, maximale Baumtiefe: 10) verwendet. Potentielle Bäume werden aus der Kombination der geomorphologischen Objekte von *r.trees.peaks*, den Farbkanälen und den verschiedenen normalisierten Differenzen abgeschätzt. Konkret wird dabei für jedes geomorphologische Objekt anhand der Schwellenwerte geschätzt, ob es tatsächlich ein Baum ist. Für die Schwellenwerte für NDVI, NIR, Gefälle und Mindestfläche gilt, je höher diese Werte, desto mehr falsche, aber auch potentielle Bäume werden ausgefiltert. Grundsätzlich sollten möglichst wenige falsche Bäume in den Trainingsdaten sein. Dafür können auch fehlende Bäume in Kauf genommen werden, diese werden beim Anwenden des trainierten Modells oft wiedererkannt.

Im Folgenden werden die **Parameter** dieses Addons genauer aufgeschlüsselt:

Eingangsparameter:

- ***red_raster***: Name des Rasters für den Rot-Kanal
- ***green_raster***: Name des Rasters für den Grün-Kanal
- ***blue_raster***: Name des Rasters für den Blau-Kanal
- ***nir_raster***: Name des Rasters für den NIR-Kanal (nahes Infrarot)
- ***ndvi_raster***: Name des NDVI-Rasterdatensatzes
- ***ndwi_raster***: Name des NDWI-Rasterdatensatzes. NDWI ist der normalisierte Wasser-Index (grün - NIR) / (grün + NIR). Dieser Input ist optional und wird intern berechnet, wenn nicht gegeben.
- ***ndgb_raster***: Name des NDGB-Rasterdatensatzes. NDGB ist die normalisierte Grün-Blau Differenz (grün - blau) / (grün + blau). Dieser Input ist optional und wird intern berechnet, wenn nicht gegeben.
- ***ndsm***: Name des nDOM-Rasterdatensatzes
- ***slope***: Rasterkarte mit dem Gefälle des nDOMs
- ***nearest***: Rasterkarte mit der ID des nächstgelegenen Baumgipfels

- ***peaks***: Rasterkarte mit geomorphologischen Gipfeln und Bergrücken

Parameter für das Erstellen der Trainingsdaten:

- ***ndvi_threshold***: NDVI-Schwellenwert, default 130
- ***nir_threshold***: NIR-Schwellenwert, default 130
- ***ndsm_threshold***: nDOM-Schwellenwert, default 1
- ***slopep75_threshold***: Aus dem nDOM wird das Gefälle berechnet und für jeden potentiellen Einzelbaum das 75%-Perzentil des Gefälles berechnet. Dieses 75%-Perzentil wird genutzt, um flache Dächer von Bäumen zu unterscheiden; default 70.
- ***area_threshold***: Mindestfläche für Einzelbäume in m², default 5

Ausgangsparameter:

- ***group***: Name der zu erstellenden Rastergruppe, in der alle für das Machine-Learning-Modell benötigten Rasterkarten zusammengefasst werden; default „ml_input“
- ***save_model***: Name der Datei, in der das Random-Forest-Modell gespeichert wird

Für die Konfiguration der Parallelisierung können folgende Parameter gesetzt werden:

- ***memory***: Arbeitsspeicher in MB, der dem Addon zur Verfügung stehen soll (optional, Default-wert ist 300 MB)

Beispiel für den Aufruf von *r.trees.mltrain*:

```
# Erstellen von Trainingsdaten und Trainieren eines Random-Forest-Modells
# mit r.trees.mltrain

# r.trees.mltrain mit Standard-Einstellungen ausführen:
$ r.trees.mltrain red_raster=top_red_02 green_raster=top_green_02 blue_ras-
ter=top_blue_02 nir_raster=top_nir_02 ndvi_raster=top_ndvi_02 ndsm=ndsm
slope=ndsm_slope nearest=nearest_tree peaks=tree_peaks group=ml_input
save_model=gelsenkirchen_2020_ml_trees_randomforest.gz
```

Hinweis:
Wir empfehlen, die Datei *save_model* immer eindeutig zu benennen, z. B. wie das *Mapset*, in dem gerechnet wird. Die Datei wird auf der ersten Ebene in der GRASS-Datenbank (*grassdb*) gespeichert und muss mit *--overwrite* überschrieben werden, wenn sie neu angelegt werden soll.

Die Standardwerte für die verschiedenen Schwellenwerte führen wahrscheinlich nicht immer zu den gewünschten Ergebnissen, was vor allem an Unterschieden in den verwendeten Luftbildern liegen kann. Dabei kann es vorkommen, dass entweder zu wenig Bäume (falsch negative) oder zu viele Bäume (falsch postive) erkannt werden. Die wichtigsten Schwellenwerte sind dafür ***ndvi_threshold*** und ***nir_threshold***. Diese sollten herabgesetzt werden, wenn mit Luftbildern gearbeitet wird, die während der trockenen Sommermonate aufgenommen wurden.

Die eigentliche Baumklassifikation mit Machine Learning erfolgt mit dem Addon *r.trees.mlapply*. Dies ist der zeitaufwendigste Schritt in der Einzelbaumerkennung und wird daher parallelisiert durchgeführt. Im Folgenden werden die **Parameter** dieses Addons genauer aufgeschlüsselt:

Eingangsparameter:

- ***area***: Name des Vektors für das aktuelle Gebiet (wie mit *m.import.rvr* importiert), default „study_area“
- ***group***: Name der mit *r.trees.mltrain* erstellten Rastergruppe, z.B. „ml_input“
- ***model***: Name der Datei, in der das Random Forest-Modell gespeichert wurde

Ausgangsparameter:

- ***output***: Name des Ergebnis-Rasters mit der Baumklassifikation

Für die Konfiguration der Parallelisierung können folgende Parameter gesetzt werden:

- ***tile_size***: Kantenlänge der Gitterkacheln für das Tiling in m, default 1000 m
- **nprocs**: Anzahl der Kerne für Parallelprozessierung (optional, Defaultwert ist -2 (Anzahl der verfügbaren Kerne minus 1), für serielle Prozessierung auf 1 setzen)

Beispiel für den Aufruf von *r.trees.mlapply*:

```
# Anwenden des trainierten Random-Forest-Modells mit r.trees.mlapply

$ r.trees.mlapply area=study_area group=ml_input \
   model=gelsenkirchen_2020_ml_trees_randomforest.gz \
   output=tree_pixels
```

Hinweis:
Dieser Schritt kann viele Stunden in Anspruch nehmen. Es ist sinnvoll, entweder die maximale Anzahl der verfügbaren Kerne minus 1 anzugeben (Default für *nprocs*), um die Rechenzeit gering zu halten.

Das Zusammenfassen der klassifizierten Baumpixel zu Einzelbäumen erfolgt mit dem Addon *r.trees.postprocess*. Als Input werden neben der Baumklassifikation von *r.trees.mlapply* die Ergebnisse der geomorphologischen Analyse mit *r.trees.peaks* verwendet. Im Folgenden werden die **Parameter** dieses Addons genauer aufgeschlüsselt:

Eingangsparameter:

- ***tree_pixels**:* Name des Rasters mit der Baumklassifikation
- ***green_raster***: Name des Rasters für den Grün-Kanal
- ***blue_raster***: Name des Rasters für den Blau-Kanal
- ***nir_raster***: Name des Rasters für den NIR-Kanal (nahes Infrarot)
- ***ndvi_raster***: Name des NDVI-Rasterdatensatzes

- ***ndwi_raster***: Name des NDWI-Rasterdatensatzes. NDWI ist der normalisierte Wasser-Index (grün - NIR) / (grün + NIR). Dieser Input ist optional und wird intern berechnet, wenn nicht gegeben.
- ***ndgb_raster***: Name des NDGB-Rasterdatensatzes. NDGB ist die normalisierte Grün-Blau Differenz (grün - blau) / (grün + blau). Dieser Input ist optional und wird intern berechnet, wenn nicht gegeben.
- ***ndsm***: Name des nDOM-Rasterdatensatzes
- ***slope***: Rasterkarte mit dem Gefälle des nDOMs
- ***nearest***: Rasterkarte mit der ID des nächstgelegenen Baumgipfels
- ***peaks***: Rasterkarte mit geomorphologischen Gipfeln und Bergrücken

Parameter für das Zusammenfassen der Baumpixel zu Einzelbäumen:

- ***ndvi_threshold***: NDVI-Schwellenwert, default 130
- ***nir_threshold***: NIR-Schwellenwert, default 130
- ***ndsm_threshold***: nDOM-Schwellenwert, default 1
- ***slopep75_threshold***: Aus dem nDOM wird das Gefälle berechnet und für jeden potentiellen Einzelbaum das 75%-Perzentil des Gefälles berechnet. Dieses 75%-Perzentil wird verwendet, um flache Dächer von Bäumen zu unterscheiden; default 70
- ***area_threshold***: Mindestgröße für Einzelbäume in m², default 5

Ausgangsparameter:

- ***trees_raster****:* Name des Ergebnis-Rasters mit Einzelbäumen
- ***trees_vector****:* Name des Ergebnis-Vektors mit Einzelbäumen

Für die Konfiguration der Parallelisierung können folgende Parameter gesetzt werden:

- ***memory***: Arbeitsspeicher in MB, der dem Addon zur Verfügung stehen soll (optional, Defaultwert ist 300 MB)

Beispiel für den Aufruf von *r.trees.postprocess*:

```
# Zusammenfassen der klassifizierten Baumpixel zu Einzelbäumen
# mit r.trees.postprocess

# r.trees.postprocess mit Standard-Einstellungen ausführen:
$ r.trees.postprocess tree_pixels=tree_pixels green_raster=top_green_02
blue_raster=top_blue_02 nir_raster=top_nir_02 ndvi_raster=top_ndvi_02
ndsm=ndsm slope=ndsm_slope nearest=nearest_tree peaks=tree_peaks trees_ras-
ter=tree_objects trees_vector=tree_objects
```

Auch hier führen die Standardwerte für die verschiedenen Schwellenwerte wahrscheinlich nicht immer zu den gewünschten Ergebnissen, was vor allem an Unterschieden in den verwendeten Luftbildern liegen kann. Auch hier sollten die Schwellenwerte ***ndvi_threshold*** und ***nir_threshold*** herabgesetzt werden, wenn mit Luftbildern gearbeitet wird, die während der trockenen Sommermonate aufgenommen wurden.

2.2.2 Ableitung diverser Baumparameter

Für die erkannten Einzelbäume können folgende Parameter abgeleitet werden: Stammposition, Höhe des Baumes, Kronendurchmesser, Kronenvolumen, Kronenfläche, NDVI aus den Farbinformationen je Einzelbaum, Abstand zum nächsten Gebäude und Abstand zum nächsten Baum. Für die Bestimmung dieser Parameter kann das Addon *v.trees.param* genutzt werden. Dieses erwartet folgenden Input:

- ***treecrowns***: Name der Einzelbaum-Vektorkarte (Polygon + Attributtabelle mit cat-Attributspalte)
- ***ndom***: Name der nDOM-Rasterkarte
- ***ndvi***: Name der NDVI-Rasterkarte
- ***buildings***: Name der Hausumringe-Vektorkarte
- ***treeparamset:*** Legt fest, welche Parameter für die einzelnen Bäume berechnet werden sollen. Die Optionen sind: position, hoehe, dm, volumen, flaeche, ndvi, dist_geb, dist_baum (ohne Leerzeichen und mit Kommas getrennt angeben). Default: Alle Parameter werden berechnet.

Optional können Parameter für die Berechnung der Distanz-Parameter gesetzt werden. Diese geben an, in welchem Umkreis nach benachbarten Gebäuden bzw. Bäumen gesucht werden soll. Insbesondere für den Abstand zum nächsten Baum beeinflusst dies die Prozessierungszeit.

- ***distance_building***: Suchradius in Metern, in dem nach dem nächsten Gebäude gesucht wird. Per default ist dieser Wert nicht gesetzt, das heißt es gibt keine Limitierung beim Suchradius und alle Gebäude werden mit einbezogen.
- ***distance_tree***: Abstand in Metern, um den die Region um den aktuellen Einzelbaum in jede Richtung vergrößert wird. In dieser vergrößerten Region wird dann nach dem nächsten Baum gesucht. Defaultwert ist 500 m.

Für die Konfiguration der Parallelisierung können folgende Parameter gesetzt werden:

- ***memory***: Arbeitsspeicher in MB, der für die Segmentierung zur Verfügung stehen soll (optional, Defaultwert ist 300 MB)
- ***nprocs*:** Anzahl der Kerne für Parallelprozessierung (optional, Defaultwert ist -2 (Anzahl der verfügbaren Kerne minus 1), für serielle Prozessierung auf 1 setzen)

Es muss kein Output definiert werden. Nach Anwenden des Addons werden die verschiedenen Parameter an die Attributtabelle des ***treecrowns-***Input geschrieben. Im Unterschied zum Massenschwerpunkt liegt der Zentroid immer innerhalb des Polygons einer Baumkrone. Die Spalten werden wie folgt benannt:

- ***pos_mass_x:*** X-Koordinate der Stammposition, bestimmt mit dem Massenschwerpunkt
- ***pos_mass_y:*** Y-Koordinate der Stammposition, bestimmt mit dem Massenschwerpunkt
- ***pos_cent_x:*** X-Koordinate der Stammposition, bestimmt mit dem Zentroid
- ***pos_cent_y:*** Y-Koordinate der Stammposition, bestimmt mit dem Zentroid
- ***height_max:*** Höhe des Baumes, bestimmt mit Maximal-Wert des nDOMs innerhalb des Einzelbaum-Polygons
- ***height_p95:*** Höhe des Baumes, bestimmt mit dem 95%-Perzentil des nDOMs innerhalb des Einzelbaum-Polygons
- ***diameter:*** Kronendurchmesser, berechnet mit Hilfe des Umfangs des Einzelbaum-Polygons und unter der Annahme einer kreisförmigen Baumkrone
- ***volume:*** Kronenvolumen, berechnet mit Hilfe des zuvor berechneten Durchmessers und unter der Annahme einer kugelförmigen Baumkrone
- ***area_sqm:*** Kronenfläche des Einzelbaum-Polygons
- ***ndvi_av:*** NDVI pro Einzelbaum, bestimmt mit Mittelwert pro Einzelbaum-Polygon
- ***ndvi_med:*** NDVI pro Einzelbaum, bestimmt mit Median pro Einzelbaum-Polygon
- ***dist_bu:*** Abstand zum nächsten Gebäude; „0“ bei direkt angrenzend an Gebäude, „NULL“ wenn kein Gebäude im Suchumkreis.
- ***dist_tree:*** Abstand zum nächsten Baum; „0“ bei direkt angrenzend an Baum, „NULL“ wenn kein Baum im Suchumkreis.

Beispiel für den Aufruf von *r.trees.param*:

```
# v.trees.param mit Standardwerten ausführen:
$ v.trees.param ndom=ndsm ndvi=top_ndvi_02 buildings=reference_buildings
treecrowns=tree_objects


# v.trees.param mit verändertem Such-Umkreis für Distanz-Parameter
$ v.trees.param ndom=ndsm ndvi=top_ndvi_02 buildings=reference_buildings
treecrowns=tree_objects distance_tree=500 distance_building=500
```

Hinweis:
Dieser Schritt kann viele Stunden in Anspruch nehmen. Es ist sinnvoll, entweder die maximale Anzahl der verfügbaren Kerne minus 1 anzugeben (Default für *nprocs*), oder keine Angabe für *nprocs* zu machen, um die Rechenzeit gering zu halten.

2.2.3 Klassifizierung der Bäume in Laub- und Nadelbäume

Die erkannten Einzelbäume können mit Hilfe des Addons *v.trees.species* in Laub- und Nadelbäume unterteilt werden. Das Addon unterteilt die Einzelbäume anhand der TOP-Helligkeitswerte, dafür müssen die folgenden Input-Parameter gesetzt werden:

Eingangsparameter:

- ***treecrowns***: Name der Einzelbaum-Vektorkarte (Polygon + Attributtabelle mit cat-Attributspalte)
- ***red_raster:*** Name der Rasterkarte des roten Kanals der TOPs
- ***green_raster:*** Name der Rasterkarte des grünen Kanals der TOPs
- ***blue_raster:*** Name der Rasterkarte des blauen Kanals der TOPs
- ***nir_raster:*** Name des Rasters für den NIR-Kanal (nahes Infrarot) der TOPs
- ***ndvi:*** Name des NDVI-Rasterdatensatzes
- ***ndsm***: Name des nDOM-Rasterdatensatzes

Optional können folgende Schwellenwerte angepasst werden, um die Klassifikation in Laub- und Nadelbäume zu beeinflussen:

- ***ndwi:*** Name des NDWI-Rasterdatensatzes. NDWI ist der normalisierte Wasser-Index (grün - NIR) / (grün + NIR). Dieser Input ist optional und wird intern berechnet, wenn nicht gegeben.

Für die Konfiguration der Parallelisierung können folgende Parameter gesetzt werden:

- ***memory***: Arbeitsspeicher in MB, der für die Segmentierung zur Verfügung stehen soll (optional, Defaultwert ist 300 MB)

Es muss kein Output definiert werden. Nach Anwenden des Addons werden die verschiedenen Parameter an die Attributtabelle des ***treecrowns*-**Input geschrieben. Die Spalten werden wie folgt benannt:

- ***species:*** Baumart (Laubbaum: "deciduous" oder Nadelbaum: "coniferous")
- ***speciesINT:*** Integer Wert für die Baumart (Laubbaum: 1 oder Nadelbaum: 2)

Beispiel für den Aufruf von *r.trees.species*:

```
# v.trees.species mit Standardwerten ausführen:
$ v.trees.species red_raster=top_red_02 green_raster=top_green_02 \
   blue_raster=top_blue_02 nir_raster=top_nir_02 ndvi=top_ndvi_02 \
   ndsm=ndsm treecrowns=tree_objects
```

2.3 Veränderungsdetektion

Die Veränderungsdetektion von zwei Einzelbaumlayern verschiedener Zeitpunkte erfolgt mit dem Addon *v.trees.cd*. Dieses berechnet drei verschiedene Vektorkarten: deckungsgleiche bzw. unveränderte Bäume, „verschwundene“ Bäume und neu erkannte Bäume.

Beide Einzelbaum-Vektorkarten müssen im aktuellen *Mapset* vorliegen. Für beide Einzelbaum-Vektorkarten müssen vorher Parameter mit *v.trees.param* berechnet worden sein. Eine Unterscheidung in Laub- und Nadelbäume ist nicht erforderlich.

Soll die Veränderungsdetektion in einem neuen *Mapset* durchgeführt werden, wird dieses zunächst erstellt und anschließend die beiden zu vergleichenden Einzelbaumlayer importiert:

```
# Erstellen eines neuen Mapsets „gelsenkirchen_cd_2020_2022“ in der Location „location_25832“
$ grass -c /grassdb/location_25832/gelsenkirchen_cd_2020_2022

$ DFOLDER=/mnt/data
$ v.in.ogr input=${DFOLDER}/2020_Sommer/gk_einzelbaeume_2020.gpkg output=trees_2020
$ v.in.ogr input=${DFOLDER}/2020_Sommer/gk_einzelbaeume_2022.gpkg output=trees_2022
```

Folgender Input für die Ausführung des Addons *v.trees.cd* wird benötigt:

- ***inp_t1***: Name der Einzelbaum-Vektorkarte zum Zeitpunkt t1
- ***inp_t2***: Name der Einzelbaum-Vektorkarte zum Zeitpunkt t2

Optional können folgende Parameter gesetzt werden:

Für deckungsgleiche/ unveränderte Bäume:

- ***vec_congr_thr***: Grenzwert (in Prozent) für Überlappung. Wenn die Überlappungsfläche der Baumkronen von t1 und t2 größer ist als die Fläche der Baumkrone zum Zeitpunkt t1 multipliziert mit diesem Grenzwert, gelten die Baumkronen als deckungsgleich. Andernfalls wird dieses Überlappungs-Polygon aus der Output-Vektorkarte entfernt. Default: 90.

Für „verschwundene“ und neue Bäume:

- ***vec_diff_min_size***: Minimalgröße der zu detektierenden Objekte in m² (Defaultwert ist 0,25 m²). Kleinere Objekte werden aus den Output-Vektorkarten entfernt.
- ***vec_diff_max_fd***: Maximalwert der *fractal dimension* (*fd*) der detektierten Objekte. Die *fractal dimension* berechnet sich als fd=2*(log(Umfang)/log(Fläche)) und ist ein Kompaktheitsmaß. Ein Maximalwert hilft, sehr lange dünne Objekte auszuschließen. (Defaultwert ist 2,5, dieser wurde empirisch ermittelt).

Zudem muss der Basisname für die drei Output-Vektorkarten definiert werden:

- ***output***: Basisname für den Output. Deckungsgleiche Bäume: ***<output>_congruent,*** „verschwundene“ Bäume: ***<output>_only_<inp_t1>***, neue Bäume: ***<output>_only_<inp_t2>***

Für die Konfiguration der Parallelisierung können folgende Parameter gesetzt werden:

- ***tile_size***: Kantenlänge der Gitterkacheln für das Tiling in m (optional, Defaultwert ist 1000 m)
- ***nprocs***: Anzahl der Kerne für Parallelprozessierung (optional, Defaultwert ist -2 (Anzahl der verfügbaren Kerne minus 1), für serielle Prozessierung auf 1 setzen)

Beispiel für den Aufruf von *v.trees.cd*:

```
# v.trees.cd mit einer Kantenlänge für das Tiling von 500 m
$ v.trees.cd inp_t1=treecrowns_2020 inp_t2=treecrowns_2022 out-
put=cd_2020_2022 tile_size=500

# v.trees.cd mit expliziten Werten für das Filtern der Output-Vektorkarten
$ v.trees.cd inp_t1=treecrowns_2020 inp_t2=treecrowns_2022 out-
put=cd_2020_2022 vec_congr_thr=80 vec_diff_min_size=1 tile_size=500
```

Hinweis:
Dieser Schritt kann viele Stunden in Anspruch nehmen. Es ist sinnvoll, entweder die maximale Anzahl der verfügbaren Kerne minus 1 anzugeben (Default für *nprocs*), um die Rechenzeit gering zu halten.

Für die Veränderungsdetektion sollte die GRASS-Datenbank auf einem lokalen schnellen Speicher liegen und nicht auf einem Netzwerkspeicher. Je nach Konfiguration des Netzwerkspeichers kann diese Analyse sehr lange dauern (über einen Monat), während sie auf einem lokalen Speicher in wenigen Tagen erfolgreich durchgeführt werden kann. Die Veränderungsdetektion braucht außerdem recht viel Platz für temporäre Daten, für eine Fläche von rund 50 km² beläuft sich das Datenvolumen auf ca. 300 GB.

2.4 Accuracy Assessment Gelsenkirchen

Für das Testgebiet Gelsenkirchen liegt als Validierungsdatensatz das Baumkataster der Stadt Gelsenkirchen vor. Dieses enthält Einzelbäume als Punkte und folgende Attribute: *BaumID, Rechtswert, Hochwert, Kartenblatt, Baumnummer, Kontrolle, Straße, Hausnummer, Baumart, Baumart_dt, Umfang, Pflanzjahr, Kronendurchmesser, Baumhöhe, Alleebau* und *Baumalter*. Die mit dem Baumanalyse-Tool erzeugten Einzelbäume liegen als Polygone mit entsprechenden Attributen vor (siehe Abschnitt 2.1.2).

Für die Validierung der erkannten Einzelbäume sowie die Unterscheidung in Laub- und Nadelbäume können die *Completeness* und *Correctness* wie folgt berechnet werden:

$$Completeness = \frac{Anzahlkorrektklassifizierter Bäume}{Anzahl Bäume Referenzdaten} * 100$$

$$Correctness = \frac{Anzahlkorrektklassifizierter Bäume}{Anzahl durch ML klassifizierte Bäume} * 100$$

Für die Laubbaum- und Nadelbaum-Unterscheidung kann zusätzlich die *Overall Accuracy* über beide Klassen berechnet werden:

$$OverallAccuracy = \frac{korrektklassifizierteLaubbäume + korrektklassifizierteNadelbäume}{AnzahlBäumeReferenzdaten} * 100$$

Für den Fall, dass mehr als ein Referenzbaum in einem der ML-klassifizierten Baum-Polygone liegt, wird dies aktuell als ein korrekt identifizierter Baum gewertet.

Für die Berechnung der Correctness müssen die Referenzdaten vollständig sein. Das Maß kann damit falsch positiv klassifizierte Bäume detektieren. Der Referenzdatensatz, in diesem Fall das Baumkataster der Stadt Gelsenkirchen, ist allerdings unvollständig. Daher wird die Berechnung der Correctness folgendermaßen modifiziert:

$$AdditionalTrees = \frac{(1 - Anzahlkorrektklassifizierter Bäume)}{(AnzahldurchMLklassifizierte Bäume)} * 100$$

Das Maß *Additional Trees* gibt damit an, wie viele der durch Machine Learning (ML) klassifizierten Bäumen nicht im Referenzdatensatz enthalten sind. Dies kann zum einen durch falsch positiv klassifizierte Bäume entstehen, zum anderen aber auch durch nicht kartierte Bäume im Baumkataster.

Zusätzlich können die Attribute der korrekt detektierten Bäume validiert werden. Dies erfolgt mit dem *RMSE* (*root mean square error, Wurzel der Abweichungsquadrate*) pro Attribut.

$$RMSE = \sqrt{\frac{\sum(berechnetesAttribut - ReferenzAttribut)^2}{AnzahlReferenzbäume}}$$

Für den Fall, dass mehr als ein Referenzbaum in einem der ML-klassifizierten Baum-Polygone liegt, wird die Fehlerdifferenz für jeden dieser Referenzbäume mit demselben ML-klassifizierten Baum berechnet.

2.4.1 Validierungsergebnisse für Gelsenkirchen 2020

Einzelbaumerkennung:

Completeness = 72,1 %

Additional Trees = 90,8 %

Es wurden ca. 72 % der Bäume aus dem Baumkataster erfasst. Dazu muss erwähnt werden, dass mit Punkt in Polygon validiert wurde. Das bedeutet, wenn ein Punkt aus dem Baumkataster etwas neben dem ML-klassifiziertem Polygon liegt, wird dieser schon als nicht richtig klassifiziert gewertet.

Ca. 91 % der klassifizierten Bäume sind nicht im Referenz-Baumkataster enthalten. Ein Teil kann theoretisch durch falsch klassifizierte Bäume (false positive) entstehen. Der Rest sind Bäume, die im Baumkataster fehlen.

Laub-Nadelbaum Unterscheidung:

Overall accuracy = 95,4 %

Laubbaum:

Completeness = 99,3 %

Correctness = 96,1 %

Nadelbaum:

Completeness = 15,3 %

Correctness = 51,5 %

Die Klassifizierung kann durch das Trainieren mit mehr Trainingsdaten weiter verbessert und generalisiert werden.

Baumparameter:

Die Ergebnisse für die einzelnen Baumparameter können Tabelle 2 entnommen werden.

Tabelle 2: Validierung der Baumparameter für Gelsenkirchen (2020)

Baumkataster	**ML-Klassifizierung**	**RMSE**
Kronendurchmesser	Kronendurchmesser (Dm)	7,7 m
Baumhöhe	Baumhöhe mit nDOM Maximum (hoe_max)	5,1 m
Baumhöhe	Baumhöhe mit nDOM 95. Perzentil (hoe_perc95)	5,0 m
Rechtswert	Rechtswert mit Zentroid (pos_zent_x)	2,5 m
Rechtswert	Rechtswert mit Massenschwerpunkt (pos_msp_x)	2,3 m
Hochwert	Hochwert mit Zentroid (pos_zent_y)	3,1 m
Hochwert	Hochwert mit Massenschwerpunkt (pos_msp_y)	2,3 m
Fläche (Annahme Kreisform, abgeleitet von Kronendurchmesser)	Fläche (tatsächliche Fläche des Polygons)	168,3 m²

Es sollte erwähnt werden, dass die Baumkronenfläche in den gegebenen Referenzdaten nicht direkt enthalten ist, sondern unter der Annahme eines Kreises bei gegebenem Kronendurchmesser berechnet wurde. Bei den ML-klassifizierten Bäumen wurde die tatsächliche Fläche des Baumpolygons berechnet.

Insbesondere bei den Parametern *Kronendurchmesser*, *Baumhöhe* und *Fläche* ist außerdem der Zeitpunkt der Erfassung des Baumkatasters (Zeitpunkt nicht bekannt) und der Input-Daten (Gelsenkirchen 2020) zu beachten.

Weiterhin ist die Definition des Kronendurchmessers entscheidend für die Validierung. Mit dem Modul *v.trees.param* wurde der Kronendurchmesser mit Hilfe des Umfangs des Einzelbaum-Polygons und unter der Annahme einer kreisförmigen Baumkrone berechnet. Die genaue Erhebung des Kronendurchmessers für die Validierungsdaten ist nicht bekannt. Bei Abweichungen kann dies daher Einfluss auf die Höhe des Fehlers haben.

Hinweise:

Neben den oben genannten Einschränkungen für die Validierung ist zu erwähnen, dass die Validierungsdaten für Gelsenkirchen unvollständig und teilweise sehr vereinfacht sind. Die Maße der Validierung beinhalten daher nicht nur tatsächliche Fehler der Methode, sondern auch Fehler der Referenzdaten.

Abbildung 1: Ausschnitt aus Gelsenkirchen mit Validierungsdaten aus dem Baumkataster Gelsenkirchen (rote Punkte)

Abbildung 2.1 zeigt einen Beispielausschnitt der Validierungsdaten. Im Baumkataster sind nur die Bäume der Allee erfasst und diese alle mit derselben Baumhöhe und demselben Durchmesser abgeschätzt. Höhere und überlagernde Bäume sind nicht enthalten. Mit der in diesem Projekt entwickelten ML-Methode wird der höhere Baum (mit entsprechender Baumhöhe und Kronendurchmesser) zusätzlich, separat erfasst. In der Validierung wird dieser mit den niedrigeren, kleineren Allee-Bäumen verglichen. Dies führt zu einem hohen Fehler, der durch Unvollständigkeiten im Baumkataster zu Stande kommt.

2.5 Gültigkeitsbereich der Analysetools

Wie jedes Klassifizierungstool weisen auch die Einzelbaum-Analysetools Grenzen auf. Grundsätzlich gilt, dass eine Klassifikation auf die Qualität ihrer Eingangsdaten angewiesen ist. Sind dort Fehler oder Ungenauigkeiten enthalten, wirkt sich dies auf das Ergebnis der Analyse aus. Im Folgenden werden der mögliche Anwendungsbereich sowie Limitierungen der Tools aufgeführt.

Zu den typischen Problemquellen bei der ***Objekterkennung*** zählen zum einen falsch positive Ergebnisse, also das Erkennen von baumartigen Objekten wie z. B. Gebäude als Bäume. Hier kann über die Schwellenwerte für die pixel- und objektbasierte Filterung im Post-Processing etwas gegengesteuert werden. Ebenso können beispielsweise Solarpanele auf Dächern als Bäume fehlinterpretiert werden, da diese im NDVI, der Höhe im nDOM und ihrer Größe den Kriterien für die Baumerkennung entsprechen können. Falls Gebäude als Bäume erkannt werden, kann dies am ehesten über den NDVI- und den NIR-Schwellenwert reguliert werden.
Darüber hinaus kann der NDVI-Schwellenwert, der sowohl zur Erstellung der Trainingsdaten als auch zur Filterung der Klassifikationsergebnisse genutzt wird, je nach genutzten TOPs variieren. Hier kommt es z. B. auf den Befliegungszeitraum für die Luftbildaufnahmen an; ggf. kann es sinnvoll sein, mit verschiedenen NDVI-Schwellenwerten zu experimentieren und/ oder große Untersuchungsgebiete in kleinere Einheiten zu unterteilen.

Das Ergebnis der ***Veränderungsdetektion*** zwischen zwei Einzelbaumlayern kann neben ganzen Bäumen, die nur in einem der beiden Layer vorkommen, auch „Randstücke“ oder „-(teil)ringe“ von Bäumen aufweisen. Diese werden durch die Defaulteinstellungen für die Mindestgröße (*min_size*) und den Maximalwert der Fractal Dimension (*max_fd*) bereits zum Teil entfernt. Es ist ggf. sinnvoll, hier mit anderen Werten zu experimentieren, um mehr dieser Objekte zu entfernen (Achtung: dadurch werden möglicherweise auch Polygone entfernt, die im Ergebnis gewünscht sind).

3 Visualisierung und Export

3.1 Visualisierung in GRASS

Die Ergebnisse von den Einzelbaum-Analysetools lassen sich in GRASS GIS visualisieren. In dem GRASS GIS-Docker ist die Graphische Oberfläche (GUI) leider nicht verfügbar, jedoch kann die

GRASS-GUI außerhalb des Dockers zur Visualisierung genutzt werden. Hierzu muss gegebenenfalls noch der Besitz der GRASS GIS ***Location*** angepasst werden (s. Kapitel 1.3.3). Dann kann die GRASS-Session gestartet werden.

```
$ grass /pfad/zu/grassdb/location_name/mapset_name
```

Falls die GRASS-GUI sich nicht mit öffnet, kann sie mit

```
$ g.gui
```

gestartet werden. Zur einfacheren Interpretation ist es hilfreich, sich im Hintergrund das TOP-Mosaik als Echtfarbenkomposit anzeigen zu lassen. Zunächst sollte der Kontrast des TOP-Mosaiks zur Visualisierung angepasst werden:

```
$ i.colors.enhance red=top_red_02 green=top_green_02 blue=top_blue_02
```

Durch Klick auf die Schaltfläche „V*erschiedene Rasterkartenlayer hinzufügen*“ - „*RGB-Karte hinzufügen*“ können die Kanäle für die Anzeigefarben <red>, <green> und <blue> gewählt werden:

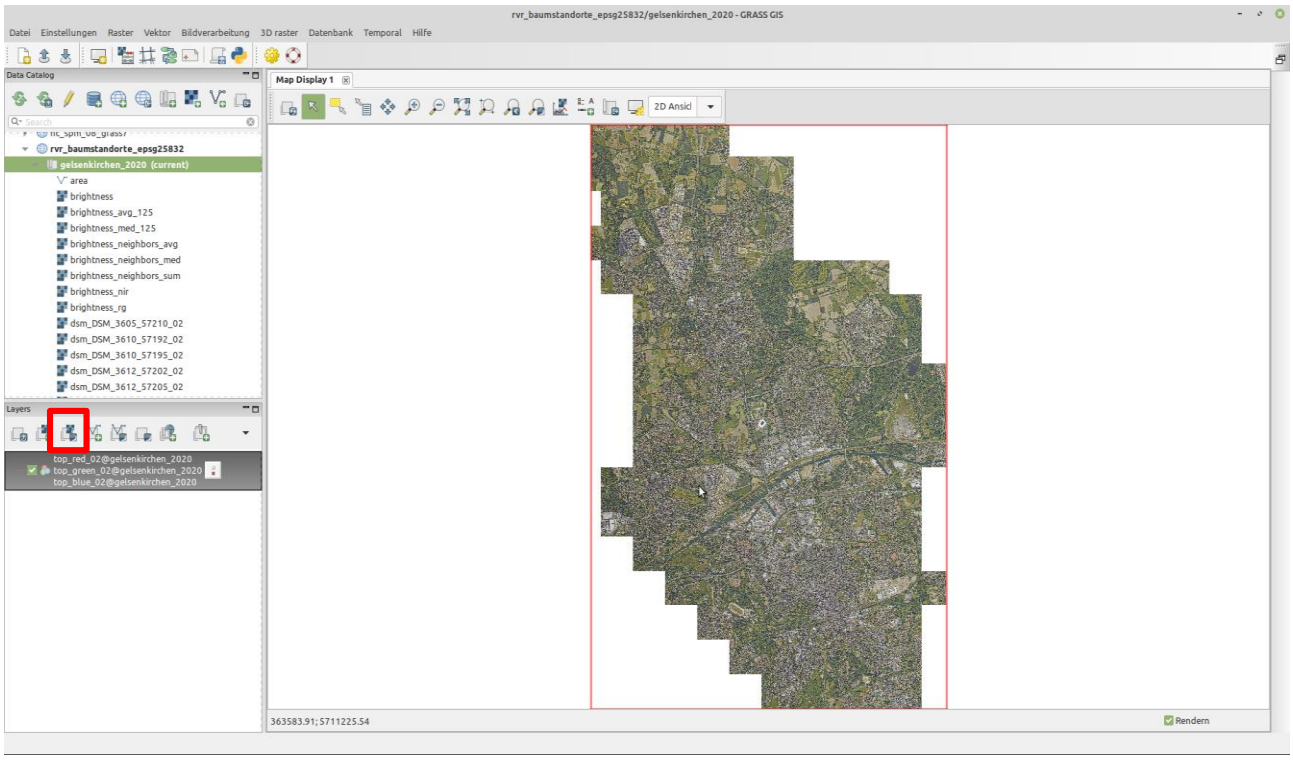

Abbildung 2: RGB-Darstellung des TOP-Mosaiks

Nun können auch die Vektor-Ergebniskarten aus *r.trees.postprocess* per Klick auf die Schaltfläche „*Vektorkarten hinzufügen*“ oder per Doppelklick auf die entsprechende Karte im *Data Catalog*-Fenster in die Ansicht geladen werden:

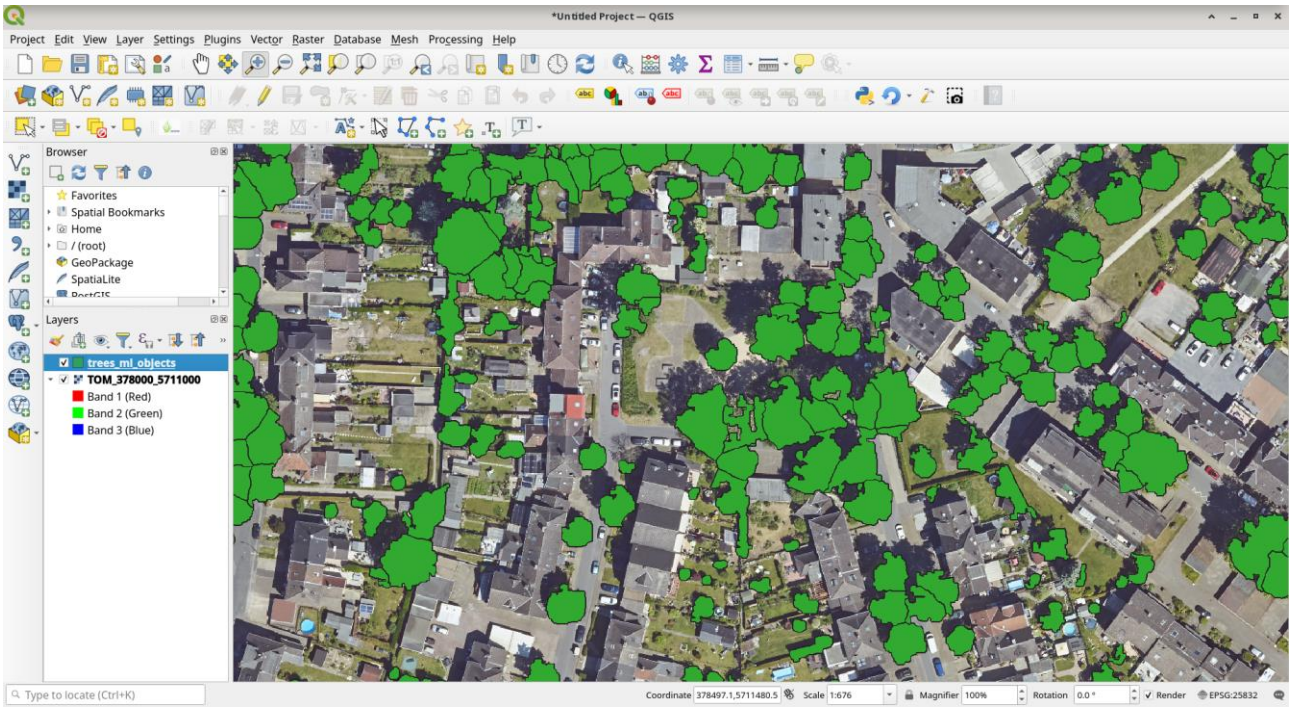

Abbildung 3: Darstellung des Einzelbaum-Ergebnislayers

3.2 Export der Daten

Für einen ersten Eindruck genügt die Visualisierung in GRASS GIS, die schnelle Anpassung der Darstellung ist jedoch in anderen Geoinformationssystemen wie QGIS oder ArcGIS etwas verständlicher. Zu diesem Zweck sowie zur weiteren Verarbeitung können die Ergebnisse daher zuletzt als gängige Vektordatenformate exportiert werden. Dazu kann das Tool *v.out.ogr* verwendet werden. Dabei muss beachtet werden, dass der Pfad zu den gemounteten Verzeichnissen (z. B. „*/mnt/data*“) angegeben werden muss, wenn im GRASS-Docker gearbeitet wird, damit die Daten nach Schließen des Dockers nicht verloren gehen. Die Dateinamen sollten zur späteren Wiedererkennung und Wiederverwendung den Namen des Gebietes und das Jahr enthalten. Über den ***format***-Parameter kann das gewünschte Dateiformat der ***output***-Datei gewählt werden:

```
# Ergebnis der Einzelbaumdetektion als Shapefile exportieren:
$ v.out.ogr input=tree_objects output=/pfad/zum/output.shp format=ESRI_Shape-
file

# Ergebnis der Veränderungsdetektion als Geopackage exportieren:
$ v.out.ogr input=cd_2020_2022 output=/pfad/zum/output_cd.gpkg format=GPKG
$ v.out.ogr input=tree_objects_2020 output=/pfad/zum/output1.gpkg format=GPKG
$ v.out.ogr input=tree_objects_2022 output=/pfad/zum/output2.gpkg format=GPKG
```

4 Umgang mit GRASS Warnungen und Fehlern

Die GRASS-Module geben prinzipiell eher häufig Warnungen aus, um eine eventuelle Fehlersuche zu erleichtern. Diese dienen der Vollständigkeit und können daher normalerweise ignoriert werden. Sollte jedoch ein Modul nicht das gewünschte Ergebnis produzieren, lohnt es sich, die GRASS-

Ausgabe auf eventuelle Warnungen hin zu überprüfen. Eine gute Anlaufstelle für mehr Informationen sind außerdem die Handbuch-Seiten der jeweiligen Module. Die Modul-Übersichten, geordnet nach Gruppen (*vector*, *raster*, *general*, etc.), können z. B. über diese Manual-Seite erreicht werden.

4.1 Häufige Fehler und Warnungen

Im Folgenden werden häufige Fehler und Warnungen beim Prozessieren von Daten mit GRASS GIS sowie der Umgang mit ihnen beschrieben. Darüber hinaus lohnt es sich bei Fehlern, insbesondere beim Im- und Export, auch in den Handbuch-Seiten der Module nach weiteren Optionen zu schauen, z. B. das Aktivieren von Flags oder Besonderheiten in den *NOTES* oder *EXAMPLES*.

4.1.1 Warnungen zu Packages

Beim Starten von GRASS GIS kann die folgende oder eine ähnliche Warnung auftreten:

```
$ DeprecationWarning: The distutils package is deprecated and slated for re-
moval in Python 3.12. Use setuptools or check PEP 632 for potential alterna-
tives from distutils.dir_util import copy_tree
```

Warnungen dieser Art können ignoriert werden. Sie beziehen sich auf Systemkomponenten (z. B. Python Packages), die zukünftig Änderungen erhalten werden. Dies hat keinen Einfluss auf die aktuelle Funktionstüchtigkeit von GRASS.

Speziell obige Warnung sollte bei der Nutzung von GRASS 8 (im Docker) nicht mehr auftreten, da die Kompatibilität bereits gewährleistet wurde. Generell laufen aktuelle GRASS-Versionen mit aktuellen Python-Versionen.

4.1.2 Warnungen beim Installieren von GRASS-Addons

Bei der Installation von GRASS-Addons können solche oder ähnliche Warnungen auftreten:

```
$ g.extension m.analyse.trees url=/src/grass-gis-addons/m.analyse.trees -s
...
WARNING: Unable to create '/usr/local/grass83/docs/html/grassdocs.css':
         '/usr/local/grass83/docs/html/grassdocs.css' and
         '/usr/local/grass83/docs/html/grassdocs.css' are the same file. Is
         the GRASS GIS documentation package installed? Installation
         continues, but documentation may not look right.
Compiling...
Installing...
Updating extensions metadata file...
Updating extension modules metadata file...
WARNING: No metadata available for module 'm.analyse.trees': Unable to
         fetch interface description for command '<m.analyse.trees>'.

         Details: <[Errno 2] No such file or directory:
         'm.analyse.trees'>
Installation of <m.analyse.trees> successfully finished
```

Die Warnungen zur GRASS GIS-Dokumentation während der Installation können ignoriert werden. Hier geht es um Metadaten und Dokumentation, die Funktion der Addons ist dadurch nicht eingeschränkt. Ob die Installation eines Addons erfolgreich verlaufen ist, kann an der letzten Zeile abgelesen werden.

4.1.3 Warnungen beim Nutzen der GUI

Beim Nutzen der Graphischen Oberfläche (GUI) treten mitunter folgende oder ähnliche Fehler auf:

```
$ (wxgui.py:6500): Gtk-CRITICAL * *: 13:46:27.691: gtk_box_gadget_distribute:
assertion 'size >= 0' failed in GtkScrollbar
$ (wxgui.py:6500): Gtk-CRITICAL * *: 14:38:27.866: gtk_box_gadget_distribute:
assertion 'size >= 0' failed in GtkCheckButton
$ (wxgui.py:6851): Gtk-CRITICAL * *: 11:49:20.118: gtk_box_gadget_distribute:
assertion 'size >= 0' failed in GtkSpinButton
```

Warnungen dieser Art können ignoriert werden. Die GUI funktioniert dennoch normal.

4.1.4 Warnungen zu inkorrekten Grenzen, *collapsed areas*, Flächenzentroiden, etc.

```
$ WARNUNG: Anzahl inkorrekter Grenzen: 8
WARNUNG: Vect_get_point_in_poly_isl(): collapsed area
WARNUNG: Kann den Flächenzentroiden nicht berechnen.
WARNUNG: Vect_get_point_in_poly_isl(): collapsed area
WARNUNG: Kann keinen Zentroid für die Fläche 101268 berechnen.
WARNUNG: Vect_get_point_in_poly_isl(): collapsed area
WARNUNG: Kann keinen Zentroid für die Fläche 201928 berechnen.
WARNUNG: Flächen mit Größe 0.0 ignoriert.
WARNUNG: Anzahl inkorrekter Grenzen: 7
WARNUNG: Die Datenbankverbindung für den Layer<1> ist nicht definiert
```

Treten Warnungen dieser Art beim Import von Daten auf, sollten sie **nicht** ignoriert werden. Bei der anschließenden Analyse hingegen können die Warnungen ignoriert werden, da die genutzten GRASS-Module, wie. z. B. *v.patch,* in der Regel auch eine topologische Säuberung beinhalten. Das heißt die Warnungen werden der Vollständigkeit halber ausgegeben und eventuelle Probleme dann direkt behoben. Eine Datenbankverbindung ist für GRASS-Vektorlayer nicht zwingend erforderlich, deswegen kann diese Warnung meist ignoriert werden. Für den Fall, dass Vektor-Attribute fehlen, könnte hier jedoch ein Grund dafür liegen.

4.1.5 Fehler bei der Veränderungsdetektion

Wenn die Veränderungsdetektion mit dem Addon *v.trees.cd* mit sehr vielen parallelen Prozessen, z.B. *nprocs=30*, aufgerufen wird, kann es beim Aktualisieren der Attribute zu einem Fehler kommen:

```
$ v.trees.cd inp_t1=trees_objects_2020 inp_t2=trees_objects out-
put=trees_herne_2020_2022 nprocs=30
       WARNING: Busy SQLITE db, already waiting for 10 seconds...
       [...]
       WARNING: Busy SQLITE db, already waiting for 60 seconds...
       DBMI-SQLite driver error:
       Unable to fetch:
       database schema has changed

       DBMI-SQLite driver error:
       Unable to fetch:
       database schema has changed

       ERROR: Unable to fetch data from table
       [...]
```

In diesem Fall muss die Anzahl paralleler Prozesse reduziert werden, z.B. auf 10, damit der ganze Prozess erfolgreich durchläuft.

4.1.6 Fehler beim Exportieren als Shapefile

Wenn beim Export als Shapefile der folgende Fehler o. ä. auftritt, ist der Name des Attributs zu lang und das Attribut muss umbenannt werden, sodass der Spaltenname maximal 10 Zeichen beinhaltet. Diese Vorgabe kommt von den ESRI Shapefile DBF- Tabellen-Spezifikationen.

```
$ v.out.ogr in=result_trees out=/pfad/zum/output.shp format=ESRI_Shapefile
ERROR 6: Failed to add field named 'durchmesser'
ERROR: Unable to create column <durchmesser>
```

Anschließend kann der Export z. B. mit folgendem Befehl wiederholt werden:

```
# Umbenennen der Spalte zu new_name
$ v.db.renamecolumn map=result_trees column="durchmesser,dm"

# Wiederholen des Exports
$ v.out.ogr input=result_trees output=/pfad/zum/output.shp format=ESRI_Shape-
file --o
```

4.1.7 Warnung beim Export: features without category were skipped

```
$ v.out.ogr in=result_trees out=/pfad/zum/output.gpkg
...
WARNUNG: 14 features without category were skipped. Features without
         category are written only when -c flag is given.
```

Die Warnung kann ignoriert werden. Es ist empfehlenswert, die *-c*-Flag beim Vektor-Export mit *v.out.ogr* nicht zu setzen. Bei Nutzung der *-c*-Flag wird das Ergebnis nicht das Erwartete sein, da in der *Simple Feature Definition* damit Polygone erstellt werden, die es eigentlich gar nicht gibt. Die

C*ategory* als GRASS-Feature ID ist nicht zu verwechseln mit den Attributen von Features (Polygonen). „*Features without category*“ bezieht sich auf Löcher in Polygonen. Diese haben weder eine eigene ID noch Attribute.

4.1.8 Fehler beim Export der Farbtabelle von Rasterdaten (z. B. DOM, nDOM) als GeoTiff

```
$ r.out.gdal input=ndsm_map output=/pfad/zum/output.tif format=GTiff
Using GDAL data type <Float32>
Die Eingabe-Rasterkarte enthält Zellen mit dem NULL-Wert (no-data). Der
Wert -nan wird verwendet, um NoData-Werte in der Eingabekarte zu
kennzeichnen. Sie können NoData-Wert mit dem Parameter nodata bestimmen.
ERROR 6: /pfad/zu/tiff/nDOM.tif, band 1: SetColorTable() only supported for
Byte or UInt16 bands in TIFF format.
```

Beim Export als GTiff ist es für Raster, die nicht als Byte oder UInt16 Datentyp vorliegen, empfehlenswert, die Farbregeln nicht mit zu exportieren. Dafür kann die *-c*-Flag gesetzt werden. Für maximale Kompatibilität mit anderen GIS-Programmen ist auch die *-m*-Flag empfehlenswert. Die Farbregeln können alternativ mit der *-t*-Flag in eine separate Raster-Attributtabelle geschrieben werden. Informationen zu den Flags finden sich auf der Handbuchseite von r.out.gdal.

Der Export kann z. B. mit folgendem Befehl wiederholt werden:

```
$ r.out.gdal -cmt input=ndsm_map output=/pfad/zum/output.tif format=GTiff --o
```

5 Referenzen

Hier befindet sich eine Übersicht der wichtigsten Referenzen in dieser Dokumentation. Kontaktieren Sie uns gerne bei Fragen oder Problemen.

Kontakt mundialis

Ansprechpersonen:

M. Metz	metz@mundalis.de
A. Weinmann	weinmann@mundialis.de
M. Mawad	mawad@mundialis.de
L. Krisztian	krisztian@mundialis.de
M. Eichhorn	eichhorn@mundialis.de

GitHub Repository mit Skripten und GRASS-Addons

https://github.com/mundialis/rvr_interface

GRASS GIS

GRASS Website

- GRASS GIS Homepage
- GRASS GIS Download

GRASS GIS Manual

- GRASS GIS Manual
- GRASS Basics im Manual

GRASS GIS Wiki

- GRASS GIS Wiki
- GRASS User Wiki Installation Guide

Tutorials

- GRASS GIS Workshop
- Analysing environmental data with GRASS GIS