DOKUMENTATION - Praktikum 2020

INHALTSVERZEICHNIS:

Im Ordner ***Atlas*** befinden sich alle in diesem Praktikum erstellten CESIUM und OpenLayers Beispiele für den Web-Atlas. Der gesamte Ordner ist als Github-Repository hochgeladen (https://github.com/eoVision/Atlas) und kann über https://eovision.github.io/Atlas/ als Webseite angesehen werden.

CESIUM

Im Ordner ***Atlas/CESIUM/BEISPIELE*** befinden sich alle erstellten CESIUM-Beispiele, d.h die jeweilige HTML-Datei, JavaScript-Datei und zusätzliche benötigte Daten (Bilder, GeoJSON, Symbole).

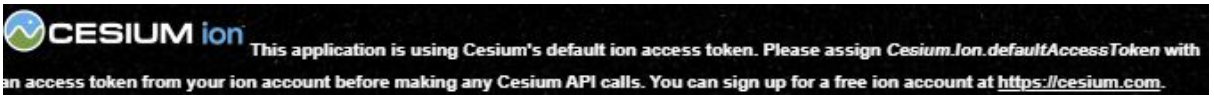
Im Ordner ***Atlas/CESIUM/CESIUMlibrary*** befinden sich die CESIUM-Bibliothek. Für die fertige CESIUM-Anwendung werden nur einige Skripte aus dieser Bibliothek benötigt und alle anderen könnten gelöscht werden.

Zum Ansehen der CESIUM-Beispiele kann die Github-Website benutzt werden (https://eovision.github.io/Atlas/). Zum lokalen Betrachten der CESIUM-Beispiele ist ein lokaler Server notwendig. Dieser kann auf dem Praktikums-PC folgendermaßen gestartet werden:

1. Eingabeaufforderung öffnen und Befehl eingeben : *python -m SimpleHTTPServer*
2. Im Browser http://localhost:8000/ eingeben und zur gewünschten HTML-Datei navigieren

Eine Anleitung zum Aufsetzen eines lokalen Servers ist hier zu finden:
https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server

Cesium ion ist ein für kommerzielle Zwecke kostenpflichtiges Angebot zum Visualisieren von Daten auf einem 3D-Globus und basiert auf der CesiumJS Bibliothek. Im Gegensatz dazu ist ***CesiumJS*** open-source und ***kostenfrei*** für alle Anwendungszwecke. In erstellten Anwendungen mit CesiumJS (https://cesium.com/cesiumjs/) ist darauf zu achten, dass keine Funktionalitäten exklusiv aus **Cesium ion** verwendet werden, da sonst ein kostenpflichtiger Access-Token notwendig ist. Wann immer dies der Fall ist, beispielsweise durch die Nutzung von geocoder, ion Base-Layern oder Terrains, wird dies in der Webanwendung angezeigt: (bzw. die Nachricht "*Upgrade for commercial use"*)



CESIUM-Beispielanwendungen mit ähnlichen Funktionen:

- Schweizer Weltatlas Globus (Zoomsets, Links): https://schweizerweltatlas.ch/virtueller-globus/
- Globus mit Links zu Webseiten: http://www.michalstefaniak.com/
- Suchfunktion/Filterfunktion: http://hiroshima.archiving.jp/index_en.htm

---

Erläuterung der CESIUM-Beispiele:

Jede der im Ordner ***Atlas/CESIUM/BEISPIELE*** liegenden HTML-Dateien ist eine funktionierende CESIUM-Applikation und greift auf darin verlinkte JavaScript-Dateien zu.

Die Datei ***GLOBUS.html*** ist ein komplexer Prototyp welcher viele der Funktionen der geplanten Anwendung beinhaltet. Zusätzlich zeigt die Anwendung ***GLOBUSinWebsite.html*** wie diese CESIUM-Funktionalitäten integriert in eine Webseite aussehen könnten und wie der Wechsel zu einer englischen Version umgesetzt werden könnte (für diesen Zweck existieren teils auch englischsprachige Versionen der Dateien mit der Endung "*EN*").

Beschreibung und Anmerkungen zu der GLOBUS-Applikation:

Die GeoJSON-Vektordateien:

Auf dem Globus sind Gebiete verzeichnet. Jedes Gebiet besteht aus einem Polygon der Gebietsgrenze und einem Punktsymbol in der Mitte des Gebietes (centroid). Die Quelle dieser Vektordaten sind geoJSON-Dateien (***centroids.geojson*** und ***polygons.geojson*** im Ordner ***CESIUM/BEISPIELE/GeoJSON***). Diese Centroid-Punkte sind notwendig um die Clustering-Funktion zu realisieren, da Polygone in CESIUM nicht geclustert werden können.

Jedem der Objekte in den geoJSON-Dateien können *properties* zugewiesen werden. In diesem Falle sind dies der Name des Gebietes, ein Link, Tags und eine Beschreibung. Auf diese Attribute kann im Quellcode zugegriffen werden und die Inhalte für Funktionen genutzt werden. Das Attribut *name* wird als Text-Label genutzt, *link* und *beschreibung* werden genutzt um die Infobox zu füllen, welche bei einem Klick des Nutzers auf eines der Objekte geöffnet wird. Das Attribut *tags* sind Kategorien nach welchen der Nutzer suchen oder filtern kann. Die Attribute (properties) der *polygons*- und *centroids*-Dateien sind inhaltlich identisch um zu erreichen dass der Nutzer sowohl auf das Polygon als auch auf das Punktsymbol klicken kann, bzw. beide Datensätze von der Suche und den Filtern beachtet werden. Diese Geojson-Dateien wurden in QGIS erstellt und die Attribute der Objekte können dort in der Attributtabelle erstellt oder geändert werden. Die Mittelpunkte der Polygone wurden in QGIS mit der Funktion *centroids* berechnet und dabei werden alle Attribute der Polygone für die erstellten Mittelpunkte übernommen, es ist also keine manuelle doppelte Eingabe notwendig. CESIUM kann lediglich Geodaten in WGS84 (EPSG: 4326) darstellen. Sollen neue Gebiete zu der Anwendung hinzugefügt werden müssen lediglich diese GeoJSON-Dateien durch aktualisierte Versionen ersetzt werden, alles weitere wird automatisch vom Quellcode erzeugt.

Die Clustering-Funktion und Zoomsets:

Zur besseren Übersichtlichkeit werden die Mittelpunkte der Gebiete geclustert. Zudem wurde eine Funktion programmiert, welche bei einem Klick auf eines der Cluster hineinzoomt um den Cluster zu öffnen und mit der Kamera zu einer detaillierteren Zoomstufe zu fliegen.
Zudem sind sowohl die Polygone als auch die Punktsymbole Teil von sogenannten *Zoomsets*. Diese bewirken, dass die Polygone erst ab einer bestimmten Kamera-Tiefe eingeblendet werden, die Punkte werden hingegen bei weitem Herauszoomen ausgeblendet.

Die Infobox:

Beim Klick auf ein Punktsymbol, Label oder Polygon der Gebiete öffnet sich eine Infobox. Diese enthält die Beschreibung, den Titel und den weiterführenden Link des Objektes, welche alle aus der GeoJSON-Datei ausgelesen werden. Der Inhalt und das Aussehen der Infobox kann mit allen zur Verfügung stehenden Möglichkeiten aus HTML verändert werden. So könnten hier die Schriftarten geändert werden, Bilder hinzugefügt werden oder gar Buttons integriert werden.
Ein Klick auf den hinterlegten Link “Link zur Anwendung” öffnet einen neuen Tab. Aus Sicherheitsgründen verbietet CESIUM das Ausführen von Skripten von diesen Seiten. Damit die verlinkten Seiten dennoch funktionieren, wurde (das einzige Mal) die Datei ***CESIUM/CESIUMlibrary/Cesium.js*** verändert und folgende Codepassage gelöscht:

c.setAttribute(“sandbox”, “allow-same-origin allow-popups allow-forms”)

Die Tags, die Suchfunktion und die Filter:

Eine Suchleiste lässt den Benutzer nach Gebieten auf dem Globus suchen. Suchtreffer bleiben sichtbar, alle anderen Objekte werden ausgeblendet. In dieser Version lässt sich nach beliebigen Zeichenfolgen suchen (auch einzelne Buchstaben) und es wird nach Treffern in den Attributen *name* und *tags* der Objekte gesucht.
Damit dem Benutzer mögliche Suchbegriffe bzw. Kategorien vorgeschlagen werden, verfügt die Suchleiste über eine autocomplete-Funktion, welche auf der bisherigen Sucheingabe des Nutzers mögliche Begriffe vorschlägt. Diese Liste der vorgeschlagenen Begriffe basiert auf den *tags* aller Objekte und wird beim Starten der Anwendung erstellt.

Zudem kann der Nutzer nach Kategorien filtern. Das Anklicken einer der Checkboxen filtert die Gebiete auf dem Globus indem geprüft wird ob diese Objekte ein bestimmten *tag* in ihrer Attributliste besitzen. Das Hinzuschalten eines zweiten Filters zeigt nur Objekte an welche die hinterlegten Tags ***beider*** Filter besitzen. Es fehlt jedoch eine Verknüpfung zwischen den angewendeten Filtern und den Suchvorschlägen der Suchleiste. Zudem wäre ein Button zum Zurücksetzen des Filters sinnvoll.

Weitere Funktionen sind:

- Ein Auswahlmenü um die verwendete Grundkarte des Globusses zu wechseln.
- Das CESIUM-Logo in der linken unteren Ecke wurde entfernt.
- Der Mauszeiger verändert sich wenn man diesen über anklickbare Elemente bewegt.

Weitere erstellte CESIUM-Anwendungen:

Zusätzlich zu GLOBUS.html wurden mehrere Anwendungen erstellt um weitere Funktionen mit CESIUM umzusetzen oder um bestimmte Teilaspekte isoliert zu zeigen:

- ***3DMarker.html*** zeigt verschiedene Wege wie Gebiete auf dem Globus markiert werden könnten. Es werden Polygone nicht aus GeoJSON-Dateien, sondern direkt im JavaScript-Code erstellt. Zudem wird gezeigt, dass man auch 3D-Modelle als Punktsymbole verwenden kann. Zudem lassen sich die Objekte auf dem Globus anklicken und führen sofort zu einem hinterlegten Link (ohne zuvor eine Infobox anzuzeigen).
- **clock.html** beinhaltet einen sich drehenden Globus und von der Sonne ausgehenden Schattenwurf. Diese Funktion wurde bei Tests jedoch auf Android-Browsern nicht korrekt dargestellt und deshalb nicht in den *GLOBUS.html*-Prototypen eingebaut.
- **Beispiele_iframe.html** zeigt wie CESIUM-Apps als iframe in eine Webseite eingebaut werden könnten. Auf beispielsweise iOS-Geräten werden hierbei jedoch die Steuerungselemente von CESIUM nicht angezeigt. Aus diesem Grund wurden die CESIUM-Fenster in *GLOBUSinWebsite.html* nicht als iframe eingebaut.
- **slider.html** zeigt wie eine beliebige Anzahl an Layern zu einem Steuerungselement hinzugefügt werden können mit welchem die Transparenz der Layer per

Schieberegler verändert werden kann. Zudem können die Layer an- und ausgeschaltet werden.

- **swipe.html** zeigt einen Swipe-Balken welcher bewegt werden kann um zwei Layer zu vergleichen
- **timeseriesczml.html** zeigt eine Zeitreihe von Ozon-Bildern. Die Animation lässt sich über ein Menü starten, stoppen und langsamer oder schneller abspielen. Zudem zeigt ein Infofenster den aktuell gezeigten Monat an.

Folgende weitere Anwendungen wurden erstellt. Diese zeigen jedoch nur isolierte Teilaspekte von bereits vorgestellten Anwendungen:

- **infoboxLink.html** zeigt wie die Inhalte der Infobox der Gebiete erstellt werden wenn keine GeoJSON-Datei verwendet wird, sondern direkt im Code erstellte Objekte.
- **Geocoder.html:** zeigt die Suchleiste
- **geojsonCluster.html:** zeigt die Cluster-Funktion
- **infoboxLinkgeojson.html:** zeigt wie die Attribute aus der geoJSON-Datei in der Infobox angezeigt werden können
- **layerpicker.html:** Fügt den Layer-Picker hinzu

Der Ordner ***Atlas/OpenLayers*** enthält sechs Beispielanwendungen. In den jeweiligen Unterordnern befinden sich alle für diese Anwendungen benötigten Dateien. Die Beispiele könne über ihre html-Datei im Browser angesehen werden. Zusätzlich befindet sich im Ordner **Atlas/*OpenLayers/ALLE BEISPIELE*** ein html-Dokument, in welchem alle sechs Beispiele auf einer Seite mittels iframe zusammengefasst sind.

OpenLayers

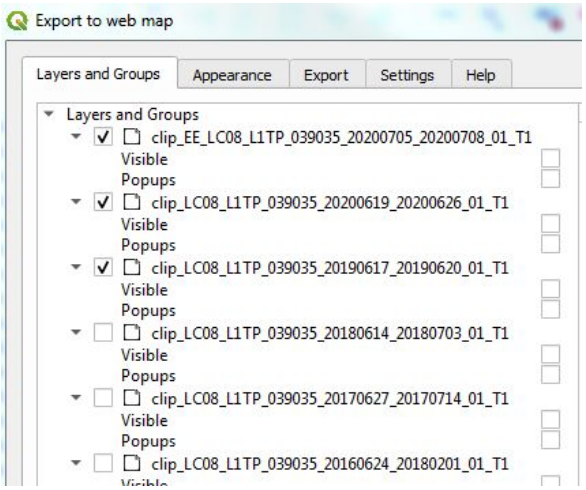
Die im folgenden erläuterten Beispiele wurden mit der JavaScript-Bibliothek OpenLayers erstellt. Zur schnellen Erzeugung von einfachen Webmaps wird das QGIS Plugin ***QGIS2Web*** verwendet. Mit diesem lassen sich QGIS Projekte in Webmaps mit Basisfunktionen umwandeln. Die Ergebnisse dieses Plugins dienen in den folgenden Beispielen als Basis und werden anschließend um individuell benötigte Funktionalitäten erweitert. Das Plugin QGIS2Web unterstützt auch die Erzeugung von Leaflet-Webmaps. Dennoch scheint OpenLayers die bessere Wahl für dieses Projekt, da eine höhere Komplexität (vor allem für Rasterdaten) mehr Möglichkeiten bieten.

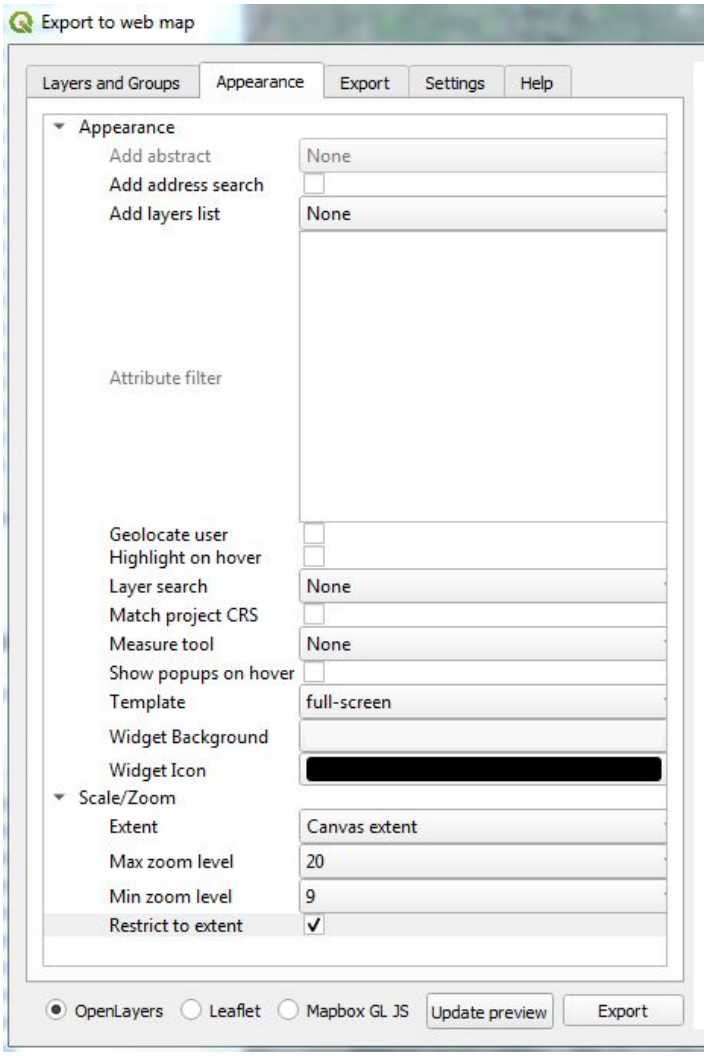
I. Webkarte mit Timeslider am Beispiel Las Vegas

(Atlas/OpenLayers/B2_QGIS2Web_TimelineLasVegas)

Die Basisfunktionen der Webmap und die eingebetteten Satellitenbilder können mit dem Plugin QGIS2Web direkt aus einem QGIS-Projekt erstellt werden.

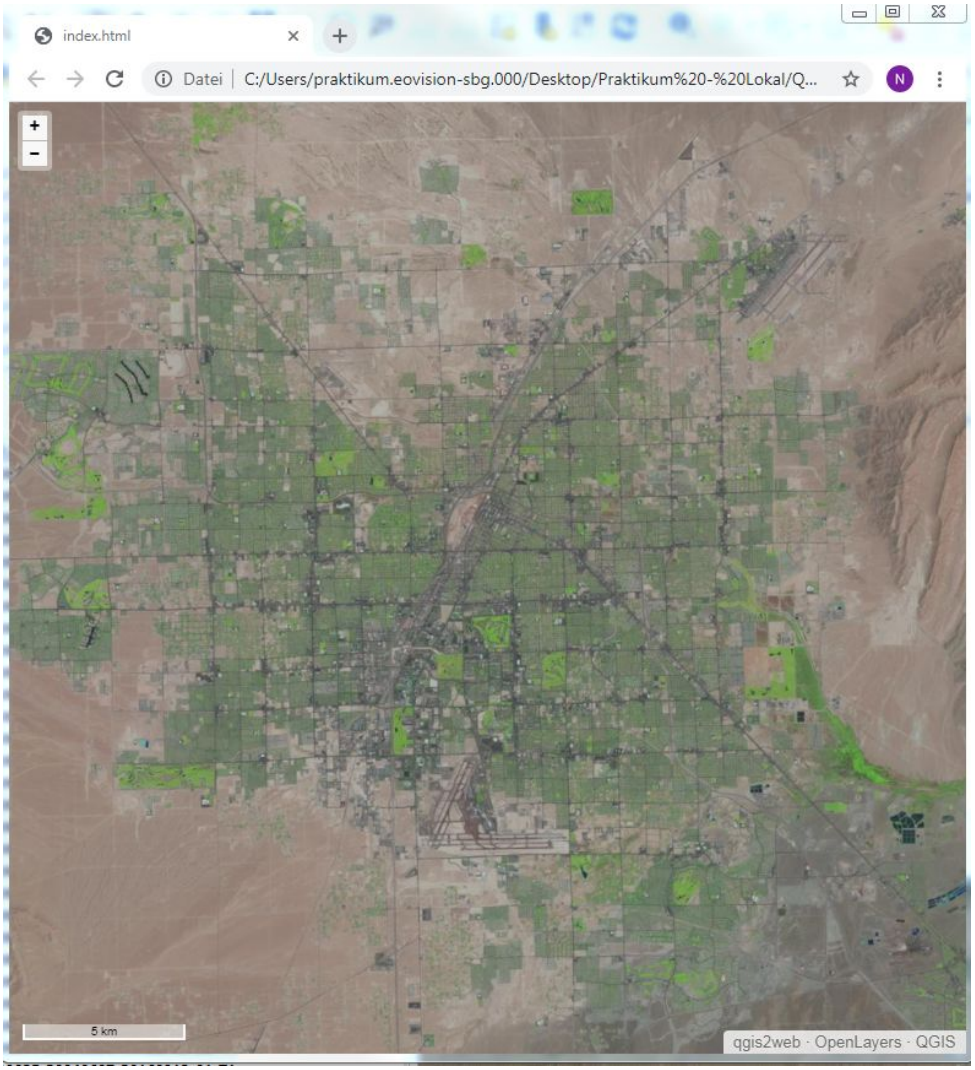
1. Alle Bilder der Zeitreihe in einem QGIS-Projekt öffnen
2. Optional: Basemap WMS wie etwa OSM hinzufügen
3. Optional: View → Decorations → Scale Bar (um in der Webmap eine Maßstabsleiste zu haben. Diese Einstellung in QGIS wird von dem Plugin QGIS2Web erkannt und umgesetzt. Alternativ lässt sich aber auch später manuell im Quellcode eine Maßstabsleiste einfach hinzufügen)
4. Web → qgis2web → Create web map (den Tab „Help“ für Erklärungen aller Funktionen des Plugins beachten)
5. Im ersten Tab des Plugins lassen sich die Layer auswählen, welche in die Webmap eingebunden werden sollen. Im Beispiel dieser Bilder-Zeitreihe ist es ratsam, dass bei jedem Layer der Haken bei „Visible“ entfernt wird, da es sonst zu Ladefehlern kommen kann, weil alle Bilder gleichzeitig beim Öffnen der Webseite geladen werden würden.





6. Im zweiten Tab des Plugins lassen sich das Aussehen und die Funktionen der Webmap einstellen. Wichtige Einstellungen für dieses Beispiel sind:
    - Add layers list: None (keinen Layerswitcher)
    - Template: full-screen (damit die Karte das gesamte Browserfenster umfasst)
    - Extent: Canvas extent (die Webkarte soll lediglich den Bereich zeigen, welche zum Zeitpunkt der Erstellung im QGIS Canvas zu sehen ist)
    - Max/Min zoom level: 20/9 (Einschränkung der Zoom-Möglichkeiten des Users)
    - Restrict to extent: ja (der User kann sich nicht aus dem definierten Bereich heraus bewegen)

7. Nach dem Export liegt eine Webkarte mit den gewünschten Funktionen vor und sieht so aus:



8. Nun kann die Webkarte mit beliebigen Funktionen von OpenLayers erweitert und angepasst werden. Die Funtkionalität des Timesliders wird durch ein OpenLayers Plugin hinzugefügt. Es ist in der Extension-Sammlung „ol-ext“ enthalten, welche hier downloadbar ist: https://github.com/Viglino/ol-ext/tree/master/dist
9. Um Extensions zu der Webkarte hinzuzufügen müssen diese in der „index.html“ Datei verlinkt werden. Hier wurden die benötigten Dateien „ol-ext.min.css“ und „ol-ext.min.js“ in den Ordner „resources“ kopiert und anschließend darauf verlinkt (vor qgis2web.js).

```
<!-- ol-ext-->
<link rel="stylesheet" href="resources/ol-ext.min.css" />
<script type="text/javascript" src="resources/ol-ext.min.js"></script>

<script src="./resources/qgis2web.js"></script>
```

10. Der Code welcher in der selbst erstellten und verlinkten Datei „resources/Anpassungen.js“ enthalten ist, greift auf die Funktionen des Plugins zu

und erstellt die Timeline. Die Funktionalität dieses Skripts ist in der Datei mit Kommentaren erklärt und benötigt für jedes Projekt individuelle Anpassungen.

Dieser Code ist eine Adaption dieses Beispiels: http://viglino.github.io/ol-ext/examples/storymap/map.control.timequake.html . Der Code wurde so angepasst, dass statt eines Vector-Datensatzes, Bilddateien als Zeitreihe dargestellt werden können.

11. Mögliche Style-Anpassungen werden mit der eingebundenen Datei „style.css“ hinzugefügt. Diese Datei ist ebenfalls mit Kommentaren versehen. Zusätzlich benötigt das Timeline Skript Zugriff auf jQuery-Code. Dieser ist ebenfalls in „index.html“ verlinkt.
12. In diesem Beispiel wurden zwei weitere Anpassungen getätigt. Die eingefügt Maßstabsleiste wurde mit einem Code in „style.css“ nach oben verschoben, sodass diese nicht mehr von der Timeline überdeckt wird.
13. Zudem wurde ein Button eingefügt, welcher die Karte im Vollbild anzeigen lässt. Dieser Code wurde in der Datei „resources/qgis2web.js“ hinzugefügt, jene Datei in welcher die eigentliche Karte und all ihre Funktionen definiert ist. Hier befindet sich auch der Code „new ol.control.ScaleLine“, welcher automatisch durch das Plugin erzeugt wird und die Maßstabsleiste zur Karte hinzugefügt.

```
var map = new ol.Map({
    controls: ol.control.defaults({attribution:false}).extend([
    //Controls Vollbild und Maßstabsleiste
        new ol.control.FullScreen(),new ol.control.ScaleLine({})
```

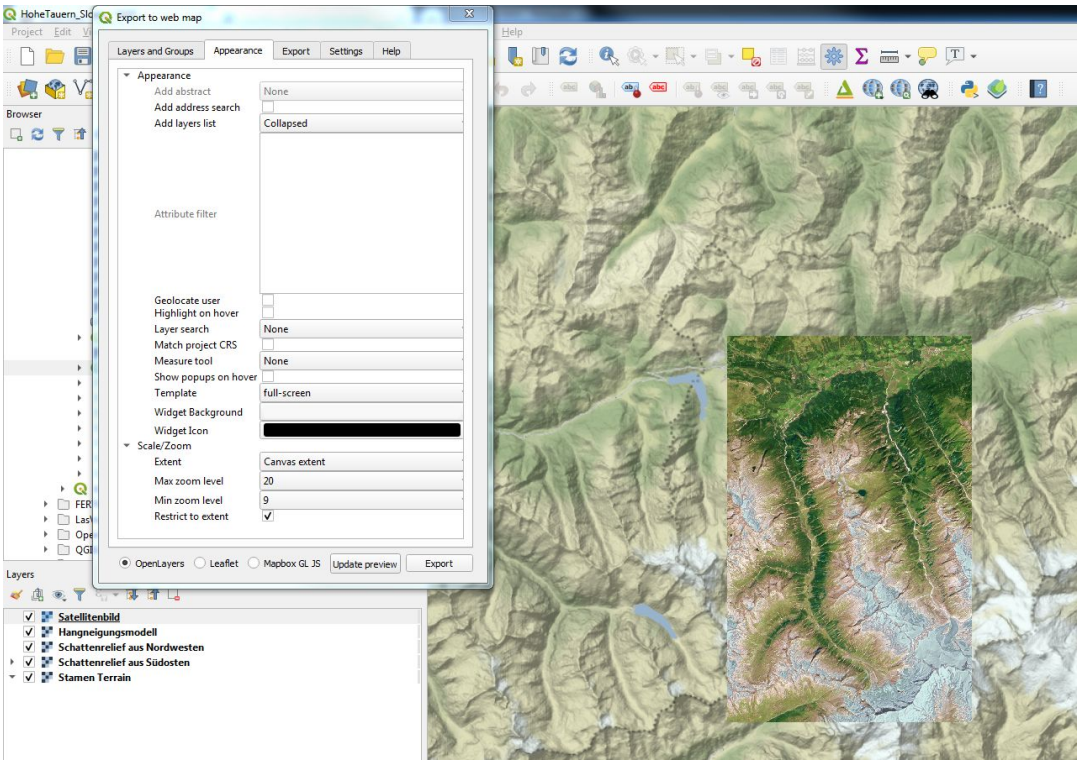
Hinweis: mit dem Plugin QGIS2Web werden die Rasterdaten aus QGIS automatisch in PNGs umgewandelt. Im Beispiel dieser Zeitreihe von LasVegas hat jedes Bild etwa 6 MB. Deshalb wurden diese Bilddateien in einem weiteren Beispiel in JPGs mit jeweils nur 0.6 MB umgewandelt. Dies verbessert die Ladezeiten enorm.

(Atlas/OpenLayers/B2_QGIS2Web_TimelineLasVegas_JPG)

II. Webkarte mit Swipe, Gitternetz und Koordinatenanzeige

(Atlas/OpenLayers/B3_QGIS2Web_SwipeGitterKoordinaten)

1. In QGIS wird ein Projekt mit allen benötigten Layern erstellt
2. Mit dem QGIS Plugin QGIS2web wird eine OpenLayers Webkarte erzeugt.



3. Das QGIS Plugin QGIS2Web fügt beim aktivieren der Option „Add layers list“ eine simple Auflistung aller in der Karte enthaltenen Layer hinzu. In diesem Beispiel wird dieser Layerswitch durch eine komplexere Layerlist ausgetauscht, welche zudem die Transparenz der Layer steuern kann und die Reihenfolge der Layer anpassen lässt. Dazu wird lediglich die verlinkung auf die Datei „resources/ol-layerswitcher.js“ gelöscht. Durch das Hinzufügen der „ol-ext“ Dateien wird die neue Version des Layerswitchers im Projekt erzeugt.

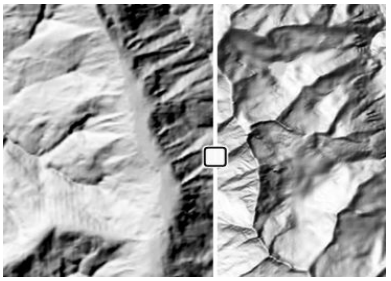
Standard:

Alternative:

4. Die erzeugte Karte wird um einige Funktionen erweitert. Eine Swipe-Funktion ist in der Extension-Sammlung „ext-ol“ enthalten (http://viglino.github.io/ol-ext/examples/storymap/map.control.timequake.html) Die CSS- und JavaScript-Dateien des Plugins werden in der „index.html“ Datei des OpenLayers Projektes verlinkt.

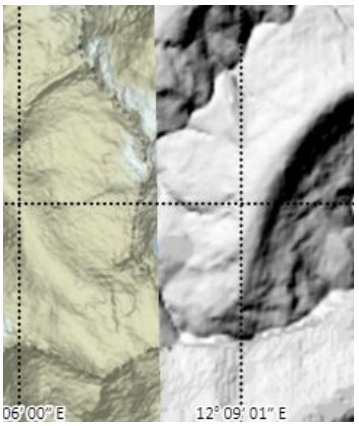
5. Als JavaScript-Code wird beispielsweise in der Datei „resources/qgis2web.js“ die Funktionalität des Swipes hinzugefügt. Die Variablennamen der benötigten Kartenlayer können in der Datei „layers/layers.js“ eingesehen werden. Durch das hinzufügen von nur 4 Zeilen ist die Swipe-Funktion in der Karte integriert.

```
// Extension: Swiper
var ctrl = new ol.control.Swipe();
map.addControl(ctrl);
// Set left
ctrl.addLayer(lyr_SchattenreliefausSdosten_1);
// Set right
ctrl.addLayer(lyr_SchattenreliefausNordwesten_2, true);
```



6. Die Karte wird um ein Gitternetz erweitert. Das standardmäßige Gitternetz von OpenLayers kann derzeit lediglich in Projekten mit dem Koordinatensystem EPSG:4326 genutzt werden. Alternativ gibt es auch experimentelle Kartennetze für andere Projektionen als Plugin (https://viglino.github.io/ol-ext/examples/canvas/map.control.graticule.html)

```
//Gitternetz
var graticule = new ol.layer.Graticule({
  // the style to use for the lines, optional.
  strokeStyle: new ol.style.Stroke({
    color: 'black',
    width: 2,
    lineDash: [0.5, 4]
  }),
      showLabels: true,
      wrapX: false
    });
graticule.setMap(map);
```



7. Des Weiteren wurde eine Anzeige eingebaut, welche die aktuellen Koordinaten an der Stelle des Mauszeigers angezeigt, ebenso durch das Hinzufügen von nur wenigen Zeilen in die „resources/qgis2web.js“ Datei.

```
//MousePositionCoordinates
var mousePositionControl = new ol.control.MousePosition({
  coordinateFormat: new ol.coordinate.createStringXY(4),
  projection: 'EPSG:4326',
  // comment the following two lines to have the mouse position
  // be placed within the map.
  //className: 'custom-mouse-position',
  //target: document.getElementById('mouse-position'),
  //undefinedHTML: ' '
});
map.addControl(mousePositionControl);
```

8. Zudem wird in der CSS-Datei „resources/qgis2web.css“ die Position und das Aussehen der Koordinatenanzeige angepasst.

```
/*Mouse Position Coordinates */
.ol-mouse-position{
    top: -moz-calc(100% - 40px);
    top: -webkit-calc(100% - 40px);
    top: calc(100% - 40px);
    left: -moz-calc(50% - 33px);
    left: -webkit-calc(50% - 33px);
    left: calc(50% - 33px);
    text-shadow: -1px 0 white, 0 1px white, 1px 0 white, 0 -1px white;
    font-weight: bold;
        }
```

III. Webmap mit kreisförmigen Spy und Nordpfeil

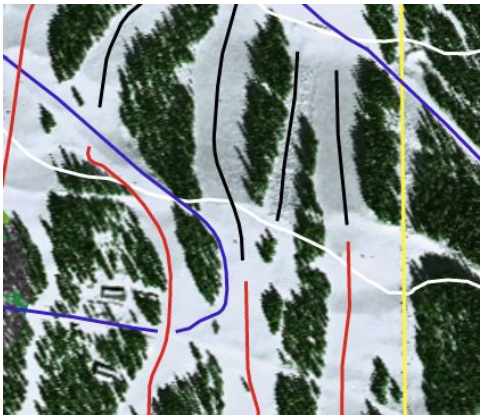
(Atlas/OpenLayers/B4_KreisNordpfeil_Skipisten)

1. Alle Layer in einem QGIS Projekt können mit dem Plugin QGIS2web zu einer Webkarte hinzugefügt werden. Bei Vektorlayern gibt es jedoch manche Einschränkungen, da nicht alle Stilelemente von QGIS in OpenLayers übersetzt werden können. In diesem Beispiel sind Höhenlinien enthalten welche in QGIS beschriftet wurden. Diese werden beim Erstellen der OpenLayers Webkarte leider nicht übersetzt.

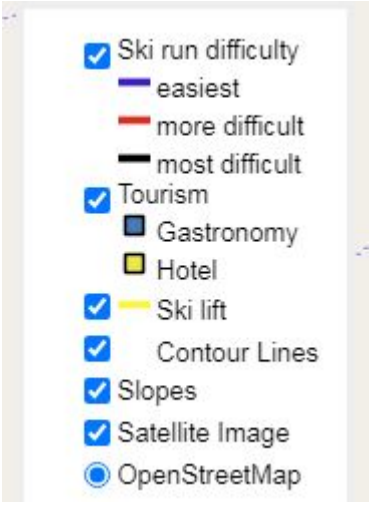
   QGIS:

   OpenLayers (keine Label):

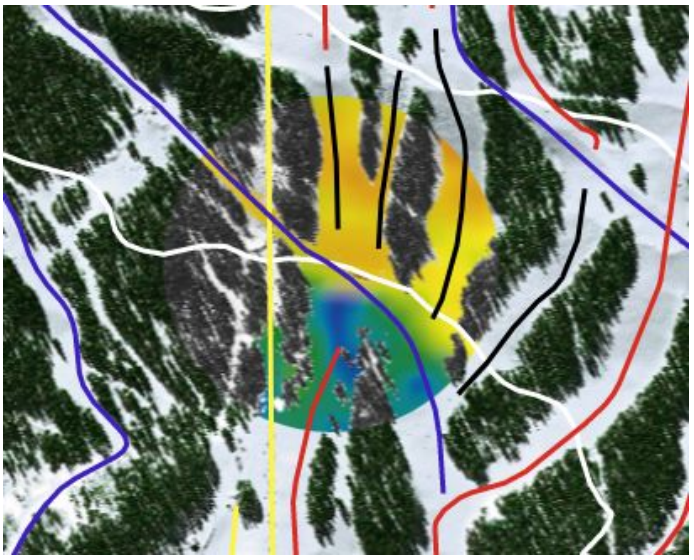
   

2. In diesem Beispiel wird in QGIS2web eine standardmäßige Layer-Liste erstellt. Diese enthält für Vektorlayer nicht nur Oberbegriffe sondern zeigt auch die Symbole der Kategorien an. Somit kann diese Layerliste auch als Legende benutzt werden.

3. Nach dem Export der Webmap aus QGIS wird in diesem Beispiel eine Funktion hinzugefügt, welche durch das Bewegen des Cursors einen im Hintergrund liegenden Layer aufdeckt und Vergleiche erlaubt. Diese Funktion ist Teil der Extension-Sammlung „ol-ext“ (https://viglino.github.io/ol-ext/examples/interaction/map.interaction.clip.html). Nach dem Verlinken der entsprechenden JS und CSS-Dateien und jQuery können die Funktionen eingebaut werden.

```
//Kreis Layerclip
var clip = new ol.interaction.Clip({ radius: Number($("#radius").val()), layers:lyr_Slopes_1 });
map.addInteraction(clip);
```



4. Eine weitere Besonderheit an diesem Beispiel ist, dass die Karte um 180° gedreht wurde und somit auf dem Kopf steht. Dies kann mit „rotation: Math.PI“ erreicht werden

```
: new ol.View({
 maxZoom: 19, minZoom: 12, rotation: Math.PI,
```

5. Zudem wurde ein Nordpfeil in die Webmap eingebaut. Dieser ist bereits in Openlayers enthalten, wurde in diesem Falle jedoch modifiziert.

```
var rotate = new ol.control.Rotate({
 autoHide: false,
 tipLabel:"nach Norden ausrichten",
});
```

```
var map = new ol.Map({
    controls: ol.control.defaults({rotate:false}).extend([
        rotate
    ]),
```

6. Zudem wurde die Position des Nordpfeils in der Webkarte in dem CSS-Dokument „resources/qgis2web.css“ angepasst

```
.ol-rotate {
left:8px;
right:unset;
top:60px;

}
```

IV. Webkarte mit alternativer Swipe-Bar

(Atlas/OpenLayers/B5_QGIS2Web_SwipeBar_Alps)

1. Die Funktionalität dieses Swipes stammt aus diesem Beispiel: https://openlayers.org/en/latest/examples/layer-swipe.html
2. Zu der mit QGIS2Web erstellten Webmap muss in „index.html“ die Swipe-Bar erstellt werden

```
<input id="swipe" type="range" style="width: 100%">
```

3. Als JavaScript-Code ist folgender Code notwendig (in resources/qgis2web.js):

```
//SWIPE
var swipe = document.getElementById('swipe');

lyr_147_03b_alps_radar_utm33_asar_1.on('prerender', function(event) {
  var ctx = event.context;
  var mapSize = map.getSize();
  var width = mapSize[0] * (swipe.value / 100);
  var tl = ol.render.getRenderPixel(event, [width, 0]);
  var tr = ol.render.getRenderPixel(event, [mapSize[0], 0]);
  var bl = ol.render.getRenderPixel(event, [width, mapSize[1]]);
  var br = ol.render.getRenderPixel(event, mapSize);

  ctx.save();
  ctx.beginPath();
  ctx.moveTo(tl[0], tl[1]);
  ctx.lineTo(bl[0], bl[1]);
  ctx.lineTo(br[0], br[1]);
  ctx.lineTo(tr[0], tr[1]);
  ctx.closePath();
  ctx.clip();
});

lyr_147_03b_alps_radar_utm33_asar_1.on('postrender', function(event) {
  var ctx = event.context;
  ctx.restore();
});

swipe.addEventListener('input', function() {
  map.render();
}, false);
```

4. Zudem muss die Größe des Webkarten-Containers in „index.html“ angepasst werden

```
<style>
html, body, #map {
    width: 100%;
    height: 700px;
```

V. Webmap mit Timeline mit Geoserver WCS als Bildquelle

(Atlas/OpenLayers/B6_TimeseriesGeoserver)

Diese Webmap basiert auf folgenden Quellen:

- Offizielle Anleitung: https://docs.geoserver.org/latest/en/user/tutorials/imagemosaic_timeseries/imagemosaic_timeseries.html
- Beispielumsetzung: https://www.earder.com/tutorials/timeseries-with-geoserver-and-openlayers/

Als Quelle für die Satellitenbilder wird in diesem Beispiel ein eigens erstellter WMS verwendet, gehostet mit Geoserver. Die oben genannten Beispiele geben detaillierte Anweisungen wie das Projekt umgesetzt werden kann. Es folgen ergänzende Anmerkungen:

1. Dieses Beispiel funktioniert nicht sofort, da dazu ein eigens gehosteter Geoserver notwendig ist.
2. In Geoserver wird ein Mosaic-Service erstellt mit den verschiedenen Bildern der Zeitreihe mit Zeitstempeln. Die Zeitstempel werden automatisch aus den Bild-Dateinamen erzeugt, die korrekte Benennung ist also wichtig. Zum Beispiel: LasVegas_20100627.tif -> 27.06.2010
3. Um diese Zeitstempel automatisch in Geoserver zu erzeugen werden zwei „.properties“-Dateien benötigt welche diese Funktion festlegen und mit den Bildern in Geoserver hochgeladen werden.
    a. indexer

```
#specifies the name of the time-variant attribute
#here the time values are stored in the column ingestion
TimeAttribute=ingestion
Schema=*the_geom:Polygon,location:String,ingestion:java.util.Date
PropertyCollectors=TimestampFileNameExtractorSPI[timeregex](ingestion)
```

    b. timeregex:

```
#this file specifies the regular expression used to extract the time information from the file name
#specifies the pattern describing the date(time) part of the filenames
#here it consists of 8 digits
regex=[0-9]{8}
```

4. Zudem müssen in der JavaScript-Datei alle Zeitstempel händisch aufgelistet werden damit die Bilder des WMS abgeglichen und abgerufen werden.

5. Eine automatische Abspielfunktion der Timeseries ist nicht enthalten, in diesem Falle wurde selbst versucht eine solche Funktion zu programmieren, diese benötigt allerdings unbedingt Verbesserungen.

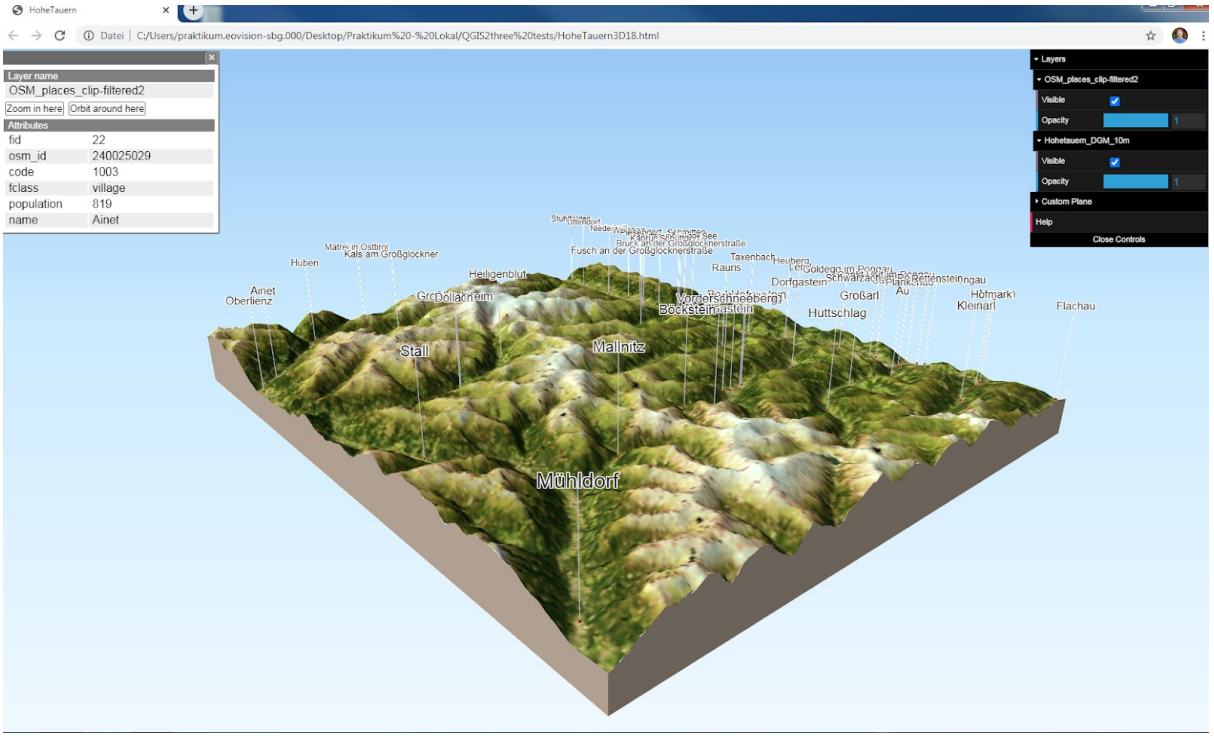
QGIS2threejs

(Atlas/OpenLayers/B1_QGIS2three_HoheTauern)

QGIS2threejs ist ein Plugin für QGIS welches die Erzeugung von in Webseiten integrierbaren 3D-Szenen ermöglicht.

Dokumentation: https://qgis2threejs.readthedocs.io/en/docs/

Beispielanwendung:
http://members.chello.at/~graser/OGDKaernten/index.html#Pfarrkirche%20Heiligenblut



Das Tool benötigt ein Höhenmodell aus welchem die 3D-Ebene erzeugt wird. Dieses 3D-Modell kann in der Erhöhung angepasst werden, in verschiedenen Qualitätsstufen ausgeben werden und durch weitere Einstellungen angepasst werden.

Das Höhenmodell wird anschließend mit einer Rasterdatei bedeckt und ist ebenfalls in der Qualität anpassbar. Zudem können Vektordateien in die Szene eingebaut werden und beschriftet werden.

In der erzeugten Szene können Features angeklickt werden und die zugehörigen Einträge der Attributtabelle werden angezeigt. In einem optionalen Kontrollmenü der Web-Anwendung können Layer an-und ausgeschaltet werden und Transparenzen angepasst werden

Vector-Layer können automatisch in 3D-Objekte umgewandelt werden (z.B. Gebäude Polygone extrudieren, Linien als Röhren, Punkte als Kugeln, u.v.m.)

Hinweise:

- Rechtsklick auf einen Layer im Plugin-Tool-Menü führt zu den Einstellungen
- Eingebundene Vektorlayer sollten auf die benötigten Maße zugeschnitten werden
- Attributfilter auf Vektorlayern werden nicht berücksichtigt, d.h. es werden trotzdem alle Features eingebunden
- Normalerweise wird der Output als AJAX-Datei ausgegeben, welche jedoch nicht lokal angesehen werden kann. Um den Output lokal betrachten zu können ist im Exportfenster „Enable the Viewer to run locally“ gewählt werden.
- Szenen mit hohem Detailgrad haben lange Ladezeiten

Einschränkungen:

- Keine Möglichkeit ein Swipe-Tool zwischen zwei Layern einzustellen
- Nur eine Rasterdatei pro Szene einfügbar
- Benutzeroberfläche nicht direkt anpassbar