ol3 Workshop

Release 3.0.0.beta2

Boundless & terrestris
A. Hocevar, B. van den Eijnden, M. Jansen & D. Koch

30.06.2014

Inhaltsverzeichnis

Herzlich Willkommen beim **OpenLayers 3 Workshop**.

Dieser Workshop soll Ihnen einen umfassenden Überblick über ol3 als Web-Mapping-Lösung geben. Die Übungen setzen voraus, dass Sie einen lokalen Webserver zur Verfügung haben. Bitte stellen Sie sicher, dass Sie die Schritte der *Vorarbeiten & Umgebung* Seite ausgeführt haben.

Dieser Workshop ist aus einer Reihe von Modulen zusammengestellt. In jedem Modul werden Sie eine Reihe von Aufgaben lösen, um ein bestimmtes Ziel zu erreichen. Jedes Modul baut iterativ Ihre Wissensbasis weiter auf.

Die folgenden Module werden in diesem Workshop behandelt:

Grundlagen

Vorarbeiten & Umgebung Vorarbeiten für den Workshop.

Basiswissen (map & view) Lernen Sie, wie man eine ol3-Karte einer Webseite hinzufügt.

Themen & Quellen (layer & source) Lernen Sie mehr über Raster- und Vektor Themen.

Benutzerinteraktion (control & interaction) Lernen Sie mehr über Interaktionen und Controls.

Fortgeschrittene Themen

Styling von Vektorlayern Lernen sie noch mehr zu Vektorlayern.

KAPITEL 1

Vorarbeiten & Umgebung

1.1 Mongoose

OpenLayers 3 läuft am besten über das HTTP Protokoll. Deshalb verwenden wir Mongoose, einen einfachen HTTP Server. Dazu laden wir von http://cesanta.com/mongoose.shtml die Free Edition von Mongoose herunter.

Außerdem werden wir uns ein Verzeichnis erzeugen, in welchem wir später die Workshopdaten einfügen und in dem Sie Ihre HTML-Seiten abspeichern können. Dazu folgen wir den Anweisungen auf http://cesanta.com/docs/BasicWebsite.shtml.

Bemerkung: Falls auf `C:\` kein Schreibzugriff möglich ist, legen Sie das `my_website` Verzeichnis am Desktop an.

1.2 ol3-Bibliothek und Demodaten

Als nächstes werden wir uns die ol3-Bibliothek und die benötigten Workshop-Daten besorgen. Laden Sie dazu die ZIP-Datei von https://github.com/bartvde/ol3-training/archive/master.zip herunter, und entpacken Sie sie im zuvor angelegten `my_website` Verzeichis.

1.3 Alles erledigt?

Gut, dann können wir nun mit *Basiswissen (map & view)* weitermachen.

Basiswissen (*map* & *view*)

2.1 Eine Karte erstellen

Eine Karte besteht in ol3 aus einer Sammlung von Themen (*layers*) und verschiedenen anderen Elementen, die Benutzerinteraktionen ermöglichen und/oder verarbeiten (*interactions* und *controls*).

Um eine Karte zu erzeugen benötigen wir drei Bestandteile:

- *HTML-Markup (Auszeichnung)*,
- *CSS-Deklarationen (Stil)* und
- *JavaScript-Initialisierungs-Code (Verhalten)*.

2.1.1 Ein lauffähiges Beispiel

Schauen wir uns zunächst ein vollständig lauffähiges Beispiel einer ol3-basierten Karte an:

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="ol3/ol.css" type="text/css">
    <style>
      #map {
        height: 256px;
        width: 512px;
      }
    </style>
    <title>OpenLayers 3 example</title>
    <script src="ol3/ol.js" type="text/javascript"></script>
  </head>
  <body>
    <h1>My Map</h1>
    <div id="map"></div>
    <script type="text/javascript">
      var map = new ol.Map({
        target: 'map',
        renderer: 'canvas',
        layers: [
          new ol.layer.Tile({
            title: "Global Imagery",
            source: new ol.source.TileWMS({
              url: 'http://maps.opengeo.org/geowebcache/service/wms',
              params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
            })
          })
        ],
        view: new ol.View2D({
```

```
          projection: 'EPSG:4326',
          center: [0, 0],
          zoom: 0,
          maxResolution: 0.703125
        })
      });
    </script>
  </body>
</html>
```

Übungen

1. Kopieren Sie obigen Text in eine neue Datei `map.html`, die Sie im Verzeichnis

   `ol3-ws/ol3-training-master`

   abspeichern.
2. Öffnen Sie die HTML-Seite in einem Browser: http://localhost/ol3-ws/ol3-training-master/map.html

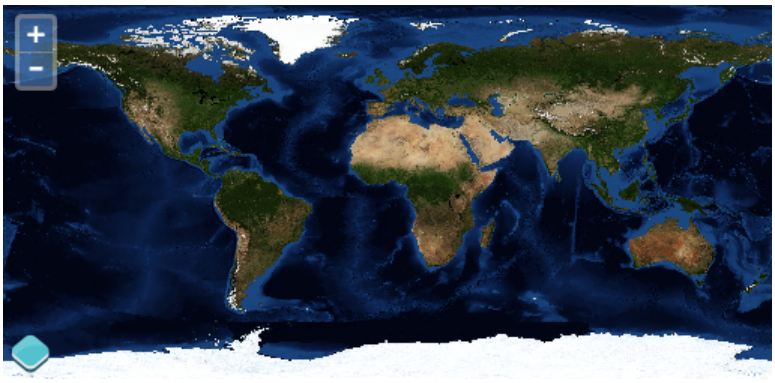


Abbildung 2.1: Die lauffähige Karte im Browser.

Nachdem wir nun eine erste funktionsfähige Karte erzeugt haben, schauen wir uns nun die *Einzelbestandteile* an.

2.2 Die Bestandteile der Karte

Wie in dem *vorherigen Abschnitt* erwähnt wurde, besteht unsere HTML-Karte aus *HTML-Markup (Auszeichnung)*, *CSS-Deklarationen (Stil)* und *JavaScript-Initialisierungs-Code (Verhalten)*. Wir werden uns jeden dieser Teile nun ein wenig genauer anschauen.

2.2.1 HTML-Markup (Auszeichnung)

Im HTML ist das Markup für die Karte recht überschaubar; wir brauchen im Grunde nur ein Elelemt:

```
<div id="map"></div>
```

Dieses `<div>`-Element wird wie ein Container für den Karten-Viewport fungieren. Wir verwenden ein `<div>`-Element, aber es kann auch jedes andere block-level-Element verwendet werden.

Wir vergeben ein `id`-Attribut, dadurch können wir es als `target` unserer Karte verwenden.

2.2.2 CSS-Deklarationen (Stil)

Zu ol3 gehört auch ein default-Stylesheet, welches festlegt, wie die einzelnen Kartenelemente gestylt werden sollen. Wir haben in unserem HTML dieses Stylesheet mittels eines `<link>`-Elements eingebunden:

```
<link rel="stylesheet" href="ol3/ol.css" type="text/css">
```

ol3 macht keine Annahmen zur Größe der Karte (bzw. des Elementes welches die Karte enthalten wird). Genau aus diesem Grunde fügen wir exakt eine weitere eigene Style-Deklaration hinzu, die die Dimensionen des Karten-`<div>`s bestimmt:

```
<link rel="stylesheet" href="ol3/ol.css" type="text/css">
<style>
  #map {
    height: 256px;
    width: 512px;
  }
</style>
```

Wir verwenden als CSS-Selektor `#map` (dies war ja die `id` des Ziel-`<div>`) und setzen die Breite (`512px`) und Höhe (`256px`) des Containers.

Diese Regel ist direkt im `<head>` der Seite innerhalb eines `<style>`-Elementes angegeben. Oft werden Sie später solche Angaben jedoch auch in einem seperaten Stylesheet notieren.

2.2.3 JavaScript-Initialisierungs-Code (Verhalten)

Der nächste Schritt zur Karte besteht aus JavaScript-Code zum Initialisieren der Karte. Wir haben hier den gesamten Code innerhalb eines `<script>`-Elements ganz am Ende des HTML `body`-Tag eingefügt:

```
<script>
  var map = new ol.Map({
    target: 'map',
    renderer: 'canvas',
    layers: [
      new ol.layer.Tile({
        source: new ol.source.TileWMS({
          url: 'http://maps.opengeo.org/geowebcache/service/wms',
          params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
        })
      })
    ],
    view: new ol.View2D({
      projection: 'EPSG:4326',
      center: [0, 0],
      zoom: 0,
      maxResolution: 0.703125
    })
  });
</script>
```

Bemerkung: Die Reihenfolge dieser Schritte ist sehr wichtig. Bevor unser benutzerdefiniertes Skript ausgeführt werden kann, muss die ol3-Bibliothek geladen werden. In unserem Beispiel wird ol3 im `<head>` unseres Dokuments mit `<script src="ol3/ol.js"></script>` geladen.

Analog hierzu kann auch der oben aufgeführte Karten-Initialisierungs-Code erst dann erfolgreich ausgeführt werden, wenn das Element, welches später die Karte enthalten wird (`<div id="map"></div>`) bereits "bereit" (sprich gerendert und Teil des Dokuments) ist. Dadurch, dass wir den `<script>`-Tag erst direkt vor dem schließenden `</body>` notieren, stellen wir sicher, dass der Karten-Container zur Verfügung steht.

Schauen wir uns die wenigen Zeilen JavaScript genauer an, um zu verstehen, wie wir die Karte erzeugen. Unser Script erzeugt eine Instanz der Klasse `ol.Map` mit einigen Konfigurations-Optionen:

`ol.Map`-Konfigurationsoption `target`

```
target: 'map'
```

Wir verwenden den Wert des `id`-Attributs unseres Karten-Containers um dem Map-Konstruktor mitzuteilen, in welches Element wir die Karte gerendert haben möchten. Wir verwenden also den String-Wert `"map"`. Diese Syntax ist eine bequeme Kurzform. Statt eines Strings kann auch eine Referenz zu dem Element angegeben werden (Solch eine Referenz kannn man etwa über `document.getElementById("map")` erhalten).

`ol.Map`-Konfigurationsoption `renderer`

Die Option `renderer` legt fest, welcher *Renderer* von ol3 verwendet werden soll. Derzeit gibt es drei verschiedene `renderer`-Typen:

- den DOM-Renderer,
- den Canvas-Renderer und
- den WebGL-Renderer.

Wir verwenden den Canvas-Renderer. Da die Bilder des WMS von einer anderen Domain (also nicht `localhost`) kommen, können wir den WebGL-Renderer nicht verwenden; dies verbietet die Same-Origin-Policy.

```
renderer: 'canvas'
```

`ol.Map`-Konfigurationsoption `layers`

Die `layers` Konfiguration erwartet eine Liste (als JavaScript-Array) aller Kartenthemen, die wir auf der Karte dargestellt haben wollen.

```
layers: [
  new ol.layer.Tile({
    source: new ol.source.TileWMS({
      url: 'http://maps.opengeo.org/geowebcache/service/wms',
      params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
    })
  })
],
```

Auf die Syntax zur Erzeugung von Layern werden wir in den folgenden Kapiteln noch näher eingehen. Lassen Sie sich also nicht von der konkreten Syntax erschrecken.

Um eine Karte sehen zu können, benötigen wir wenigstens einen Layer.

`ol.Map`-Konfigurationsoption `view`

Als letztes wollen wir den `view` (etwa *Kartenausschnitt*) konfigurieren. Wir spezifizieren eine Projektion (`projection`), ein Kartenzentrum (`center`) und den initialen Zoomlevel (`zoom`). Außerdem legen wir eine `maxResolution` fest, so dass wir keine Kartenausschnitte anfordern, die von dem entfernten Dienst (in diesem Falle ein *GeoWebCache*) nicht geliefert werden können.

```
view: new ol.View2D({
   projection: 'EPSG:4326',
   center: [0, 0],
   zoom: 0,
   maxResolution: 0.703125
})
```

Herzlichen Glückwunsch: Sie haben soeben erfolgreich Ihre erste Kartenanwendung mit ol3 programmiert, seziert und analysiert!

Um erfolgreich weiterarbeiten zu können, schauen wir uns *weitere Ressourcen* an, die Sie als ol3-Entwickler benötigen werden.

2.3 Weitere ol3 Ressourcen

ol3 stellt bereits ein große Menge an Funktionalität zur Verfügung. Obwohl die ol3-Entwickler hart gearbeitet haben, um Beispiele für viele der Funktionalitäten zu entwickeln, und der Code so auch vor dem Hintergrund einfacher Nachvollziehbarkeit organisiert und geschrieben wurde, haben viele Entwickler zunächst Schwierigkeiten auf ol3 umzusteigen.

Hier werden zukünftige Versionen von ol3 sicherlich ansetzen müssen, um noch bessere Handreichungen zu geben.

2.3.1 Lernen am Beispiel

Die meisten neuen Nutzer von ol3 lernen vermutlich am einfachsten die Verwendung der Bibliothek durch das Nachvollziehen von Beispielen. Hier kann man meist kleine spezifische Aspekte in Aktion betrachten und mit dem Code *spielen*, um ihn zu verstehen.

- http://ol3js.org/en/master/examples/

2.3.2 Thematische Dokumentation

Zu speziellen Themen gibt es bereits Prosa-Dokumentation. Die Anzahl an behandelten Themen wird sicherlich noch wachsen.

- http://ol3js.org/en/master/doc/quickstart.html
- http://ol3js.org/en/master/doc/tutorials

2.3.3 API-Dokumentation

Nachdem die Basiskomponenten, die eine `ol.Map` beeinflussen und ausmachen verstanden sind, hilft ein Blick in die aus dem Quelltext generierte API-Dokumentation. Hier finden sich Details zu Klassen, Methoden-Signaturen oder Objekteigenschaften etc.

- http://ol3js.org/en/master/apidoc/

Die API-Dokumentation ist noch im Aufbau und nicht zu 100% vollständig. Auch hieran arbeiten die ol3-Entwickler fieberhaft.

2.3.4 Teil der Community werden

ol3 wird entwickelt und gewartet von einer Gemeinschaft aus Programmierern und Benutzern wie Ihnen! Unabhängig davon, ob Sie Fragen stellen oder Code beitragen wollen, die einfachste und erste Form direkter Partizipation ist sicherlich die Registrierung auf der (englischsprachigen) Mailingliste

- https://groups.google.com/forum/#!forum/ol3-dev

Hier ist jeder neue Nutzer willkommen und keine Frage zu grundlegend!

2.3.5 Fehler melden

Wir freuen uns über Feedback auch insbesondere, wenn Sie glauben einen Bug gefunden zu haben.

Um Fehler zu melden, ist es wichtig, dass Sie die verschiedenen *Varianten* von ol3 kennen:

- `ol.js` - Diese Datei wird durch den *Google Closure Compiler* (im *Advanced Mode*) erstellt und ist für Menschen i.d.R. nicht lesbar.
- `ol-simple.js` - Diese Datei wird ebenfalls durch den *Google Closure Compiler* (jedoch im *Simple Mode*) erstellt und ist bereits eher les- und verstehbar (vgl. auch https://developers.google.com/closure/compiler/faq)
- `ol-whitespace.js` - Menschenlesbare Version von ol3, die zum Debuggen & Fehler melden verwendet werden sollte.

Wenn Sie einen Fehler feststellen, ist es wichtig, dass die Meldung des Fehlers auf Basis der `ol-whitespace.js`-Datei geschieht. Wenn möglich, sollte auch der zum Fehler führende *stack trace* Teil der Fehlermeldung sein. Ein solcher *stack trace* lässt sich etwa mittels verschiedener Browserwerkzeuge (etwa *Chrome developer Tools*) generieren.

Um dies zu testen wollen wir in `map.html` einen Fehler produzieren. Ändern Sie hierzu den Layertyp von `ol.layer.Tile` zu `ol.layer.Image`. Auf der Konsole müssten Sie die folgende Fehlermeldung erhalten:

```
Uncaught TypeError: Object #<yc> has no method 'tb'
```

Niemand kann Ihnen hierzu bei Fragen weitere Informationen oder Hilfe geben.

Wenn statt `ol.js` jedoch `ol-whitespace.js` eingebunden wird, so hält der JavaScript-Debugger des Browser nach einem Neuladen der Seite an der kritischen Stelle die Ausführung des Codes an, und ein Debugging ist deutlich vereinfacht.

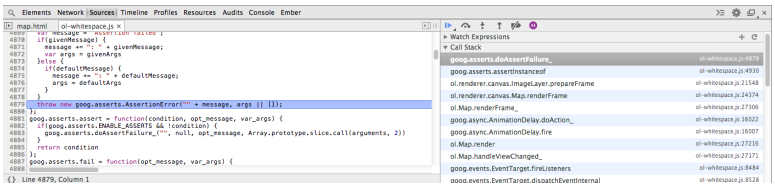


Abbildung 2.2: Der *stack trace* im Debugger. Mittels der rechten Maustaste kann jener kopiert werden.

Themen & Quellen (*layer* & *source*)

3.1 *Web Map Service*-Themen (WMS)

Wenn Sie ein Thema (`layer`) Ihrer Karte hinzufügen, ist die Themenquelle (`source`) in der Regel für das Abrufen der angezeigt Daten verantwortlich. Die angeforderten Daten können entweder Raster- oder Vektordaten sein. Rasterdaten stellen wir uns hierbei als serverseitig erzeugtes Bild vor. Vektordaten werden als strukturierte Informationen vom Server ausgeliefert, und sie werden für die Anzeige erst auf dem Client (also in ihrem Browser) gerendert.

Es gibt mannigfaltige Services, die Rasterdaten bereitstellen. In diesem Abschnitt werden wir uns mit denjenigen befassen, die Rasterdaten konform zur OGC (Open Geospatial Consortium, Inc.) Web Map Service (WMS)-Spezifikation liefern.

3.1.1 Einen WMS-Layer erzeugen

Wir werden nun zunächst wieder mit einem vollständigen Beispiel beginnen. Nach und nach werden wir die `layer` der Karte modifizieren, um ein Verständnis für die Funktionalität zu bekommen.

Schauen wir uns folgenden Code an:

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="ol3/ol.css" type="text/css">
    <style>
      #map {
        height: 256px;
        width: 512px;
      }
    </style>
    <script src="ol3/ol.js" type="text/javascript"></script>
    <title>OpenLayers 3 example</title>
  </head>
  <body>
    <h1>My Map</h1>
    <div id="map"></div>
    <script type="text/javascript">
      var map = new ol.Map({
        target: 'map',
        renderer: 'canvas',
        layers: [
          new ol.layer.Tile({
            title: "Global Imagery",
            source: new ol.source.TileWMS({
              url: 'http://maps.opengeo.org/geowebcache/service/wms',
              params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
```

```
            })
          })
        ],
        view: new ol.View2D({
          projection: 'EPSG:4326',
          center: [0, 0],
          zoom: 0,
          maxResolution: 0.703125
        })
      });
    </script>
  </body>
</html>
```

Übungen

1. Falls noch nicht geschehen, speichern Sie obigen Text als `map.html` im Workshopverzeichnis.
2. Testen Sie, ob die Karte wie erwartet funktioniert:

   http://localhost/ol3-ws/ol3-training-master/map.html

3.1.2 Der `ol.layer.Tile`-Konstruktor

Der `ol.layer.Tile`-Konstruktor wird mit einem Konfigurationsobjekt vom Typ `olx.layer.TileOptions` aufgerufen (vgl. http://ol3js.org/en/master/apidoc/olx.layer.html#TileOptions). Im vorliegenden Fall ist der Wert des `source`-Schlüsselwortes eine Instanz der Klasse `ol.source.TileWMS`. Ein lesbarer Titel kann für den Layer mit der `title`-Option festgelegt werden.

Im Gegensatz zu OpenLayers 2.x (wo die Aufgaben von *source* und *layer* gemeinsam von `OpenLayers.Layer`-Instanzen übernommen wurden), unterscheidet ol3 immer zwischen einem Thema (`layer`) und seiner Datenquelle (*source*).

Die Klasse `ol.layer.Tile` repräsentiert ein reguläres Netz oder *Grid* von Bildern. Ein *einzelnes* Bild als Thema ist durch die Klasse `ol.layer.Image` abgebildet.

Je nach Layertyp (`...Tile` oder `...Image`) sollte auch eine andere `source` verwendet werden: `ol.source.TileWMS` beziehungsweise `ol.source.ImageWMS`.

3.1.3 Der `ol.source.TileWMS`-Konstruktor

Der `ol.source.TileWMS`-Konstruktor erwartet ein einzelnes Objekt als Argument. Die Struktur dieses Objektes ist unter http://ol3js.org/en/master/apidoc/olx.source.html#TileWMSOptions aufgeführt.

Der Schlüssel `url` enthält den Link auf die *Online Resource* des Dienstes und `params` ist ein Objektliteral mit Parameternamen als Schlüssel und Parameterwerten als Wert. Die default `VERSION` eines WMS ist in ol3 `1.3.0`, falls Ihre Dienste diese Version nicht unterstützen, kann ein kleinerer Wert (etwa `1.1.1`) über die `params` eingestellt werden.

```
layers: [
  new ol.layer.Tile({
    title: "Global Imagery",
    source: new ol.source.TileWMS({
      url: 'http://maps.opengeo.org/geowebcache/service/wms',
      params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
    })
  })
]
```

Übungen

1. Der oben verwende WMS stellt auch einen Layer namens `"openstreetmap"` zur Verfügung. Ändern Sie den `ol.layer.Tile`-Konstruktor so, dass jener Layer angefragt und angezeigt wird.

   Anschließend sollte Ihr Code etwa wie folgt aussehen

   ```
   new ol.layer.Tile({
     title: "Global Imagery",
     source: new ol.source.TileWMS({
       url: 'http://maps.opengeo.org/geowebcache/service/wms',
       params: {
         // hier ist die wesentliche Änderung
         LAYERS: 'openstreetmap',
         VERSION: '1.1.1'
       }
     })
   })
   ```

2. Ändern Sie den Code dahingehend ab, dass statt vieler einzelner Kacheln nur noch ein **einzelnes** Bild beim WMS angefragt wird. Die folgende Seiten aus der API-Dokumentation können hilfreich sein: http://ol3js.org/en/master/apidoc/ol.layer.Image.html und http://ol3js.org/en/master/apidoc/ol.source.ImageWMS.html. Sie sollten die Adresse `http://suite.opengeo.org/geoserver/wms` und den Layernamen `opengeo:countries` verwenden.

3. Überprüfen Sie mittels der *Developer Tools* des Browsers, ob tatsächlich nur noch **ein** Bild angefordert wird statt der 256x256 Pixel großen Kacheln.

Abbildung 3.1: Eine Karte, die den Layer `"openstreetmap"` als `"image/png"` darstellt.

Nachdem wir erfolgreich mit dynamisch berechneten Bildern gearbeitet haben, wollen wir uns nun *vorberechneten Kacheln* zuwenden.

3.2 Vorberechnete Kacheln (*tiles*)

Standardmäßig fragt der `Tile`-Layer Bilder in 256 x 256 Pixel Größe an, um den Kartenviewport (und einen kleinen Bereich darüber hinaus) zu füllen. Während die Karte verschoben oder gezoomt wird, gehen weitere Anfragen in eben dieser Größe an den Server, um jeweils den aktuellen Bereich auszufüllen. Einige Bilder wird der Browser lokal vorhalten (*cachen*) trotzdem ist der Prozessierungsaufwand auf dem Server unter Umständen hoch, da die Bilder dynamisch berechnet werden.

Da die letztlich generierten Anfragen in einem regulären Grid deterministisch sind, kann ein Server die Anfragen ggf. vorberechnen und bei einer Anfrage nur ein hoffentlich bereits bestehendes Bild ausliefern. Üblicherweise führt dies zu einer Performancesteigerung sowohl auf der Client- als auch auf der Serverseite.

3.2.1 Die Klasse `ol.source.XYZ`

Die *Web Map Service*-Spezifikation räumt anfragenden Clients eine große Freiheit ein, die letztlich in unendlich vielen verschiedenen Requests münden. Ohne weitere Einschränkungen ist es in der Praxis daher sehr schwierig (wenn nicht unmöglich), diese Anfragen zu *cachen*.

Das gegenteilige Extrem stellt ein Service dar, der ausschließlich eine fixe Anzahl an Zoomstufen unterstützt, und in jenen auch nur solche, die in ein gewisses Grid passen. Solche Dienste lassen sich in einer XYZ-Quelle generalisieren; *X* und *Y* repräsentieren dann Spalten und Zeilen in jenem Grid und *Z* steht für die Zoomstufe.

3.2.2 Die Klasse `ol.source.OSM`

Das OpenStreetMap (OSM)-Projekt stellt vielfaltige & weltweite Geodaten frei zur Verfügung, die von Freiwilligen gesammelt und erhoben wurden. OSM stellt verschiedene gerenderte Kachel-Sets dieser Daten zur Verfügung. Da jene Kacheln analog zum *XYZ-Grid* organisiert sind, kann man sie in ol3 nutzen. Die Klasse `ol.source.OSM` greift auf diese *tiles* zu.

Übungen

1. Öffen Sie die Datei `map.html` vom *vorherigen Abschnitt* in einem Texteditor und ändern Sie die Karten-Initialisierung wie folgt:

```
<script>
  var map = new ol.Map({
    target: 'map',
    renderer: 'canvas',
    layers: [
      new ol.layer.Tile({
        source: new ol.source.OSM()
      })
    ],
    view: new ol.View2D({
      center: ol.proj.transform([-93.27, 44.98], 'EPSG:4326', 'EPSG:3857'),
      zoom: 9
    })
  });
</script>
```

2. Im `<head>` Bereich des Dokumentes fügen Sie bitte einige CSS-Deklarationen für die Copyright-Angaben (*attribution*) von ol3 hinzu.

```
<style>
    #map {
        width: 512px;
        height: 256px;
    }
    .ol-attribution ul,
    .ol-attribution a,
    .ol-attribution a:not([ie8andbelow]) {
        color: black !important;
    }
</style>
```

3. Speichern Sie Ihre Änderungen und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html

Abbildung 3.2: Eine Karte, die einen gekachelten Layer aus der OpenStreetMap Quelle darstellt.

Einige Details

Projektionen

Schauen wir uns die `view`-Definition genauer an:

```
view: new ol.View2D({
  center: ol.proj.transform([-93.27, 44.98], 'EPSG:4326', 'EPSG:3857'),
  zoom: 9
})
```

Räumliche Daten können in verschiedenen räumlichen Koordinatensystemen vorliegen. Während ein Datensatz zum Beispiel geographische Koordinaten (Längen- und Breitengrade) verwendet, kann ein anderer Datensatz in lokaler metrischer Projektion vorliegen. Eine vollständige Diskussion von Koordinatenbezugssystemen liegt sicherlich außerhalb des Fokus dieses Workshops, aber es ist wichtig, das wesentliche Konzept zu verstehen.

ol3 muss das Koordinatensystem Ihrer Daten kennen. Intern werden Projektionen von der Klasse `ol.proj.Projection` abgebildet. Die `transform` Funktion im `ol.proj` Namensraum akzeptiert Auch *Strings* die das Koordinatensystem repräsentieren (oben sind dies `"EPSG:4326"` und `"EPSG:3857"`).

Koordinatentransformierung

Die OpenStreetMap Kacheln, die wir verwenden, liegen in einer Merkator Projektion vor. Wir müssen daher auch das Kartenzentrum in Merkator-Koordinaten angeben. Da es relativ einfach ist, geographischen Koordinaten zu einem gewissen Ort auf der Welt zu bekommen, nutzen wir `ol.proj.transform` um geographische Koordinaten (`"EPSG:4326"`) in Merkator Koordinaten (`"EPSG:3857"`) umzuwandeln.

Angepasste Karten-Optionen

Bemerkung: Die Projektionen, die wir bislang verwendet haben, sind die einzigen, die ol3 per default kennt. Für andere Projektionen müssen wir etwas mehr Aufwand betreiben.

```
var projection = ol.proj.configureProj4jsProjection({
  code: 'EPSG:21781',
  extent: [485869.5728, 76443.1884, 837076.5648, 299941.7864]
});
```

Des weiteren benötigen wir zwei zusätzliche `<script>`-Tags:

```
<script type="text/javascript"
  src="http://cdnjs.cloudflare.com/ajax/libs/proj4js/1.1.0/proj4js-compressed.js">
</script>
<script type="text/javascript"
  src="http://cdnjs.cloudflare.com/ajax/libs/proj4js/1.1.0/defs/EPSG21781.js">
</script>
```

Unter http://spatialreference.org/ oder http://epsg.io/ kann man die relevanten Informationen nachschauen, sofern der EPSG-Code bekannt ist.

Erzeugung des Layers

```
layers: [
  new ol.layer.Tile({
    source: new ol.source.OSM()
  })
],
```

Wie zuvor erzeugen wir einen Layer und fügen ihn der Karten-Konfiguration hinzu. Der Konstruktor `ol.source.OSM()` wird ohne Argument aufgerufen, daher gelten für diese Instanz die ol3-Defaults.

Stil

```
.ol-attribution ul,
.ol-attribution a,
.ol-attribution a:not([ie8andbelow]) {
  color: black;
}
```

Wie ol3-Controls zu handhaben sind, steht nicht im Fokus dieses Abschnitts. Als kleine Vorschau sei jedoch hier bereits erwähnt, dass standardmäßig jede `ol.Map` eine `ol.control.Attribution` Control hat, die etwa Copyright-Informationen zu den Kartenthemen darstellt.

Obige CSS-Angaben ändern das Aussehen dieser Angaben, welche auf der Karte im unteren Bereich sichtbar sind.

Nachdem wir erfolgreich öffentlich verfügbare gekachelte *TileSets* verwendet haben, schauen wir uns an, wie wir *proprietäre Rasterthemen* einsetzen können.

3.3 Proprietäre Raster Themen

Die vorherigen Abschnitte verwendeten Layer, die auf dem standardisiertem WMS (OGC Web Map Service) oder eigenen Kachelsets basierten. Online Kartenwendungen wurden aber nicht zuletzt durch die Verfügbarkeit von proprietären Kacheldiensten so populär. ol3 stellt Layertypen zur Verfügung, die mit jenen Diensten über deren jeweilige API kommunizieren können.

In diesem Abschnitt werden wir das Beispiel aus dem *vorherigen Abschnitt* erweitern, in dem wir einen Layer hinzufügen, der die Bing-Kacheln als Quelle nutzt.

Um zu verstehen, wie ol3 zusammen mit *Google Maps* verwendet werden kann, sei auf das entsprechende Online Beispiel unter http://ol3js.org/en/master/examples/google-map.html verwiesen.

3.3.1 Bing!

Wir fügen nun also einen Bing-Layer hinzu:

Übungen

1. Suchen Sie in der Datei `map.html` die Stelle, an der die OSM (OpenStreetMap) `source` konfiguriert wird und ändern Sie jene zu einer `ol.source.BingMaps`:

```
source: new ol.source.BingMaps({
  imagerySet: 'Road',
  key: 'Ak-dzM4wZjSqTlzveKz5u0d4IQ4bRzVI309GxmkgSVr1ewS6iPSrOvOKhA-CJlm3'
})
```

Bemerkung: Die Bing-API muss mit einem API-Schlüssel `key` angesprochen werden. Das vorliegende Beispiel verwendet einen solchen Schlüssel, der nicht für Ihre eigenen Anwendungen verwendet werden sollte. Um selbst einen Schlüssel zu erhalten, kann die Registrierung unter https://www.bingmapsportal.com/ verwendet werden.

---

2. Speichern Sie ihre Änderungen in der Datei `map.html` und testen Sie das Ergebnis im Browser: http://localhost/ol3-ws/ol3-training-master/map.html

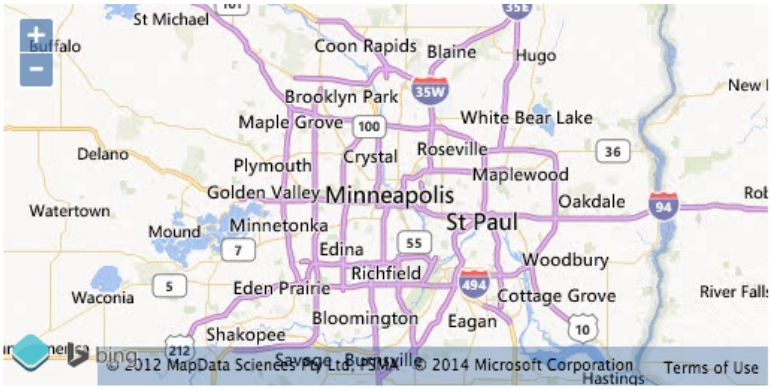


Abbildung 3.3: Eine ol3-Karte, die die Bing-Kacheln verwendet.

Vollständiges Beispiel

Die von Ihnen angepasste HTML-Datei `map.html` sollte etwa wie folgt aussehen:

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="ol3/ol.css" type="text/css">
    <style>
      #map {
        height: 256px;
        width: 512px;
      }
      .ol-attribution ul,
      .ol-attribution a,
      .ol-attribution a:not([ie8andbelow]) {
        color: black;
      }
    </style>
    <script src="ol3/ol.js" type="text/javascript"></script>
    <title>OpenLayers 3 example</title>
  </head>
  <body>
    <h1>My Map</h1>
    <div id="map" class="map"></div>
    <script type="text/javascript">
      var map = new ol.Map({
```

```
        target: 'map',
        renderer: 'canvas',
        layers: [
          new ol.layer.Tile({
            source: new ol.source.BingMaps({
              imagerySet: 'Road',
              key: 'Ak-dzM4wZjSqTlzveKz5u0d4IQ4bRzVI309GxmkgSVr1ewS6iPSrOvOKhA-CJlm3'
            })
          })
        ],
        view: new ol.View2D({
          center: ol.proj.transform([-93.27, 44.98], 'EPSG:4326', 'EPSG:3857'),
          zoom: 9
        })
      });
    </script>
  </body>
</html>
```

Als letzten Layertyp wollen wir uns *Vektorthemen* anschauen.

3.4 Vektorlayer

Vektorlayer werden durch die Klasse `ol.layer.Vector` bereitgestellt. Dieser Layertyp kümmert sich um die Darstellung von Vektordaten auf der Client-Seite.

Derzeit unterstützt in ol3 ausschließlich der *Canvas*-Renderer die Verwendung von `ol.layer.Vector`.

3.4.1 Vektorfeatures gerendert im Browser

Wir kehren zunächst mit unserer Karte zurück auf den Stand des ersten Beispiels (Ansicht der gesamten Welt). Hierauf wollen wir anschließend Vektoren darstellen.

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="ol3/ol.css" type="text/css">
    <style>
      #map {
        height: 256px;
        width: 512px;
      }
    </style>
    <title>OpenLayers 3 example</title>
    <script src="ol3/ol.js" type="text/javascript"></script>
  </head>
  <body>
    <h1>My Map</h1>
    <div id="map"></div>
    <script type="text/javascript">
      var map = new ol.Map({
        target: 'map',
        renderer: 'canvas',
        layers: [
          new ol.layer.Tile({
            title: "Global Imagery",
            source: new ol.source.TileWMS({
              url: 'http://maps.opengeo.org/geowebcache/service/wms',
              params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
            })
```

```
          })
        ],
        view: new ol.View2D({
          projection: 'EPSG:4326',
          center: [0, 0],
          zoom: 0,
          maxResolution: 0.703125
        })
      });
    </script>
  </body>
</html>
```

Übungen

1. Öffnen Sie die Datei `map.html` in einem Texteditor und fügen Sie obigen Code ein.
2. Speichern Sie ihre Änderungen in der Datei `map.html` und testen Sie das Ergebnis im Browser: http://localhost/ol3-ws/ol3-training-master/map.html
3. Im Initialisierungscode der Karte fügen Sie bitte nach dem `Tile`-Layer einen weiteren Layer hinzu. Kopieren Sie am einfachsten hierzu die folgenden Zeilen, die einen Vektorlayer definieren, der seine Daten aus einer lokalen GeoJSON-Datei bezieht:

```
new ol.layer.Vector({
  title: 'Earthquakes',
  source: new ol.source.GeoJSON({
    url: 'data/layers/7day-M2.5.json'
  }),
  style: new ol.style.Style({
    image: new ol.style.Circle({
      radius: 3,
      fill: new ol.style.Fill({color: 'white'})
    })
  })
})
```

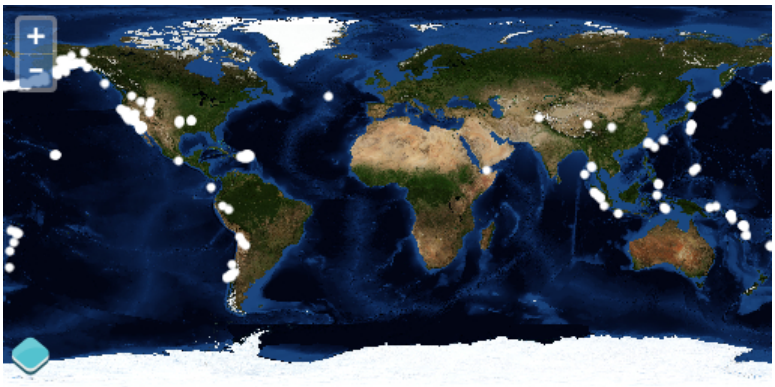


Abbildung 3.4: Eine Weltkarte mit weißen Punkten, die Erdbeben repräsentieren.

Bemerkung: Da die Daten in der GeoJSON-Datei in `EPSG:4326` vorliegen, und der `view` der Karte auch diese Projektion verwendet, müssen wir die Daten nicht reprojizieren. Falls die Datenprojektion nicht übereinstimmt, kann man bei der Konstruktion der `source` einen `projection`-Key im Konfigurationsobjekt übergeben.

Details des Beispiels

Wir schauen uns nun die Erzeugung des Vektorlayers im Detail an um zu verstehen, was im Beispiel passiert.

```
new ol.layer.Vector({
  title: 'Earthquakes',
  source: new ol.source.GeoJSON({
    url: 'data/layers/7day-M2.5.json'
  }),
  style: new ol.style.Style({
    image: new ol.style.Circle({
      radius: 3,
      fill: new ol.style.Fill({color: 'white'})
    })
  })
})
```

Der Layer wird mit dem Titel (`title`) *Earthquakes* und einigen weiteren Optionen initialisisert. Insbesondere von Relevanz ist der `source`-Schlüssel, welchem wir eine Instanz von `ol.source.GeoJSON` zuweisen. Diese `source` verweist auf die URL zur GeoJSON-Datei.

Bemerkung: Wollten wir die Features des Layers attributiv ausgestalten, so könnten wir statt einer Instanz von `ol.style.Style` auch eine Stylefunktion übergeben.

Zusatzaufgaben

1. Jeder weiße Kreis repräsentiert ein `ol.Feature` Objekt des `ol.layer.Vector`, und jedes Feature hat die Attribute `summary` & `title`. Registrieren Sie eine Funktion, die bei jedem `singleclick`-Event, welcher auf der `ol.Map` gefeuert wird, die Funktion `forEachFeatureAtPixel` der `map` aufruft und gegebenenfalls weitere Erdbebeninformationen ausgibt.
2. Die hier verwendeten Daten entstammen der Seite http://earthquake.usgs.gov/earthquakes/catalogs/ des USGS. Recherchieren Sie dort, ob Sie weitere Geodaten in einem von ol3 unterstützten Format finden und versuchen Sie, jene in `map.html` zu integrieren.

3.5 Statische Vektoren (*ImageVector*)

Wenn sowohl die Daten als auch deren Ausgestalten nicht dynamisch, sondern im Grunde *fix* sind, kann es aus Perfomancegründen Sinn machen, dass ol3 ein Bild der Vektordaten berechnet.

3.5.1 Die Klasse `ol.source.ImageVector`

Unsere Beispielkarte soll nun zunächst auf einen Stand zurückversetzt werden, in welchem die Erdbebendaten auf der Weltkarte angezeigt werden.

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="ol3/ol.css" type="text/css">
    <style>
      #map {
        height: 256px;
        width: 512px;
      }
    </style>
    <title>OpenLayers 3 example</title>
```

```
    <script src="ol3/ol.js" type="text/javascript"></script>
  </head>
  <body>
    <h1>My Map</h1>
    <div id="map"></div>
    <script type="text/javascript">
      var map = new ol.Map({
        target: 'map',
        renderer: 'canvas',
        layers: [
          new ol.layer.Tile({
            title: "Global Imagery",
            source: new ol.source.TileWMS({
              url: 'http://maps.opengeo.org/geowebcache/service/wms',
              params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
            })
          }),
          new ol.layer.Vector({
            title: 'Earthquakes',
              source: new ol.source.GeoJSON({
              url: 'data/layers/7day-M2.5.json'
            }),
            style: new ol.style.Style({
              image: new ol.style.Circle({
              radius: 3,
              fill: new ol.style.Fill({color: 'white'})
            })
          })
        })],
        view: new ol.View2D({
          projection: 'EPSG:4326',
          center: [0, 0],
          zoom: 0,
          maxResolution: 0.703125
        })
      });
    </script>
  </body>
</html>
```

Tasks

1. Öffnen Sie die Datei `map.html` in einem Texteditor und fügen Sie obigen Code ein.
2. Speichern Sie ihre Änderungen in der Datei `map.html` und testen Sie das Ergebnis im Browser: http://localhost/ol3-ws/ol3-training-master/map.html
3. Ändern Sie dann den Vektorlayer wie folgt:

```
new ol.layer.Image({
  title: 'Earthquakes',
  source: new ol.source.ImageVector({
    source: new ol.source.GeoJSON({
      url: 'data/layers/7day-M2.5.json'
    }),
    style: new ol.style.Style({
      image: new ol.style.Circle({
      radius: 3,
        fill: new ol.style.Fill({color: 'white'})
      })
    })
  })
})
```

4. Aktualisieren Sie die Seite http://localhost/ol3-ws/ol3-training-master/map.html im Browser.

Bemerkung: Wenn alles korrekt ist, sehen Sie die gleichen Daten wie zuvor, selbst *feature detection* (aus einer Zusatzaufgabe) sollte noch funktionieren. Allerdings sollten die Daten weniger scharf gerendert werden. Effektiv haben wir Perfomance zu Gunsten der Qualität des Renderings gewonnen.

Details des Beispiels

Wir werfen einen erneuten Blick auf die Erzeugung des `ol.layer.Image`.

```
new ol.layer.Image({
  title: 'Earthquakes',
  source: new ol.source.ImageVector({
    source: new ol.source.GeoJSON({
      url: 'data/layers/7day-M2.5.json'
    }),
    style: new ol.style.Style({
      image: new ol.style.Circle({
      radius: 3,
        fill: new ol.style.Fill({color: 'white'})
      })
    })
  })
})
```

Wir verwenden `ol.layer.Image` statt `ol.layer.Vector`, trotzdem können wir nach wie vor, mittels `ol.source.ImageVector`, auf die zuvor verwendete `source` vom Typ `ol.source.GeoJSON` zurückgreifen.

Anders als zuvor wird die Ausgestaltung als `style` *nicht* dem Layer sondern der `ol.source.ImageVector` mitgegeben.

Zusatzaufgabe

1. Funktioniert unser `singleclick`-Handler aus dem vorherigen Abschnitt noch?

   Zur Erinnerung:

   Registrieren Sie eine Funktion, die bei jedem `singleclick`-Event, welcher auf der `ol.Map` gefeuert wird, die Funktion `forEachFeatureAtPixel` der `map` aufruft und gegebenenfalls weitere Erdbebeninformationen ausgibt.

Benutzerinteraktion (`control` & `interaction`)

4.1 Einen Maßstabsbalken einblenden

Auf Karten werden nicht selten Maßstabsbalken zur besseren Orientierung angezeigt. Mit ol3 ist dies auch möglich: die Klasse `ol.control.ScaleLine` kann hierzu verwendet werden.

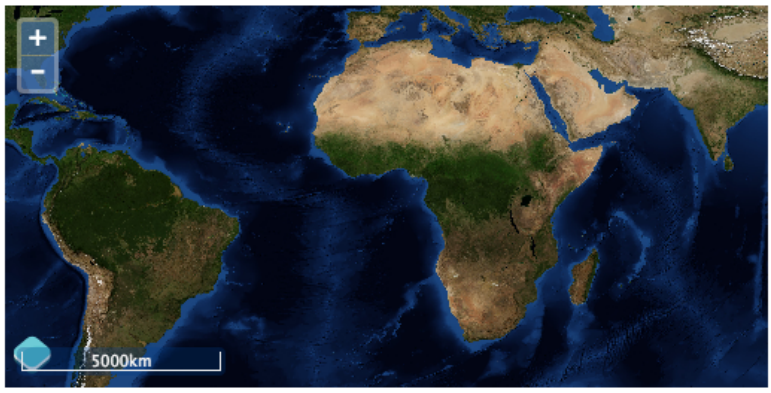
4.1.1 Erzeugung des Maßstabsbalkens

Übungen

1. Öffnen Sie `map.html` im Texteditor.
2. Fügen Sie nachfolgenden Code irgendwo innerhalb der `ol.Map`-Konfiguration hinzu:

```
controls: ol.control.defaults().extend([
  new ol.control.ScaleLine()
]),
```

3. Speichern Sie Ihre Änderungen und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html



Der Maßstabsbalken ohne Anpassungen.

4.1.2 Anpassen der `ScaleLine`-Control

Vermutlich finden auch Sie, dass der Maßstabsbalken schlecht zu lesen ist, wenn er auf der Weltkarte angezeigt wird. Man kann verschiedenen Dinge unternehmen, um die Lesbarkeit zu verbessern. Vielleicht reicht es, wenn wir den CSS-Stil anpassen? Wir wollen zunächst eine andere Hintergrundfarbe festlegen und auch etwas mehr Innenabstand zuweisen:

```
.ol-scale-line,
.ol-scale-line:not([ie8andbelow]) {
  background: black;
  padding: 5px;
}
```

Aus didaktischen Gründen wollen wir aber nun annehmen, dass Ihr Karten-`div` sowieso schon überfüllt ist, und Sie daher entschieden haben, dass der Maßstabsbalken in ein separates `<div>`-Element außerhalb der Karte plaziert werden sollte.

Hierzu werden wir zunächst ein solches Element dem HTML hinzufügen, und anschließend der `ol.control.ScaleLine` mitteilen, **wo** sie gerendert werden soll.

Übungen

1. Erzeugen sie ein neues `<div>`-Element auf Ihrer HTML-Seite. Um den Zugriff auf das Element zu vereinfachen, geben Sie dem `<div>` bitte die `id="scale-line"`. Fügen Sie z.B. den nachfolgenden Code ein:

   ```
   <div id="scale-line" class="scale-line"></div>
   ```

2. Anschließend müssen wir den Code modifizieren, der die `ScaleLine‘ erzeugt. Beachten Sie vor allem den Schlüssel ‘‘target`:

   ```
   controls: ol.control.defaults().extend([
     new ol.control.ScaleLine({
       className: 'ol-scale-line',
       target: document.getElementById('scale-line')
     })
   ]),
   ```

3. Speichern Sie Ihre Änderungen und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html

4. Was ändern die folgenden Zeilen CSS?

   ```
   .scale-line {
     position: absolute;
     top: 350px;
   }
   .ol-scale-line {
     position: relative;
     bottom: 0px;
     left: 0px;
   }
   ```

5. Speichern Sie Ihre Änderungen erneut und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html

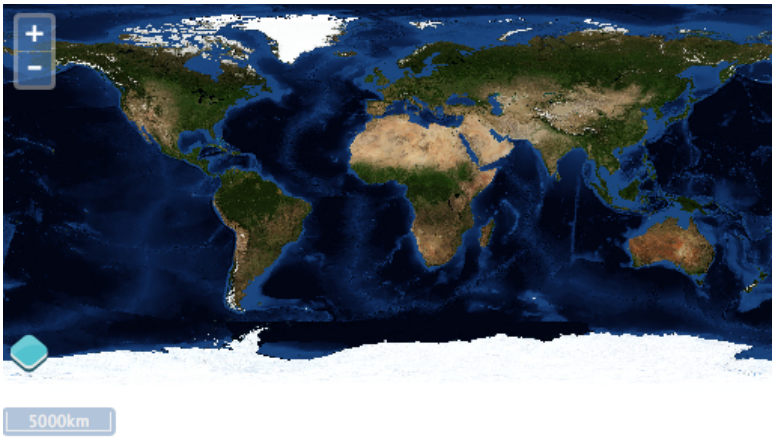


Abbildung 4.1: Ein Maßstabsbalken in separatem `<div>`-Element.

Bemerkung: Um eigene *controls* zu erzeugen sei auf http://ol3js.org/en/master/examples/custom-controls.html verwiesen.

4.2 Features selektieren

In den vorherigen Abschnitten zu Layern haben wir gelernt, das wir geographische Entitäten (*Features*) als Vektoren in ol3 laden und dynamisch auf die Karte zeichnen können. Einer der Vorteile von Vektordaten, ist die einfache Möglichkeit zur Interaktion mit jenen im Kartenclient. Wir wollen nun einen Vektorlayer erzeugen, in dem Benutzer Features selektieren können und Informationen zum ausgewählten Objekt erhalten.

4.2.1 Vektorlayer und `Select`-Interaktion erzeugen

Übungen

1. Wir beginnen mit dem lauffähigen Beispiel aus dem *vorherigen Abschnitt*. Öffnen Sie die Datei `map.html` im Texteditor und stellen Sie sicher, dass der Inhalt etwa wie folgt aussieht.

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="ol3/ol.css" type="text/css">
    <style>
    #map {
      height: 256px;
      width: 512px;
    }
    </style>
    <script src="ol3/ol.js" type="text/javascript"></script>
    <title>OpenLayers 3 example</title>
  </head>
  <body>
    <h1>My Map</h1>
    <div id="map"></div>
    <script type="text/javascript">
      var map = new ol.Map({
        interactions: ol.interaction.defaults().extend([
          new ol.interaction.Select({
            featureOverlay: new ol.FeatureOverlay({
```

```
            style: new ol.style.Style({
              image: new ol.style.Circle({
                radius: 5,
                fill: new ol.style.Fill({
                  color: '#FF0000'
                }),
                stroke: new ol.style.Stroke({
                  color: '#000000'
                })
              })
            })
          })
        })
      ]),
      target: 'map',
      renderer: 'canvas',
      layers: [
        new ol.layer.Tile({
          title: "Global Imagery",
          source: new ol.source.TileWMS({
            url: 'http://maps.opengeo.org/geowebcache/service/wms',
            params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
          })
        }),
        new ol.layer.Vector({
          title: 'Earthquakes',
          source: new ol.source.GeoJSON({
            url: 'data/layers/7day-M2.5.json'
          }),
          style: new ol.style.Style({
            image: new ol.style.Circle({
              radius: 5,
              fill: new ol.style.Fill({
                color: '#0000FF'
              }),
              stroke: new ol.style.Stroke({
                color: '#000000'
              })
            })
          })
        })
      ],
      view: new ol.View2D({
        projection: 'EPSG:4326',
        center: [0, 0],
        zoom: 1
      })
    });
  </script>
</body>
</html>
```

2. Speichern Sie Ihre Änderungen und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html

   Klicken Sie mit der linken Maustaste auf ein Feature um es zu selektieren.

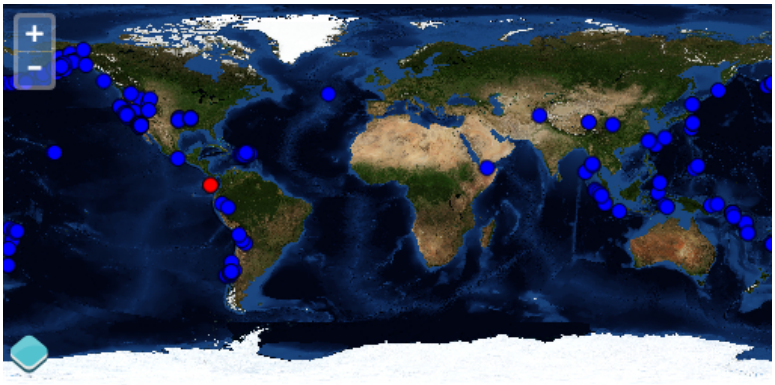


Abbildung 4.2: Featureselektion mittels einer `Select`-Interaktion.

4.3 Features neuzeichnen

Neue Features kann man mittels einer `ol.interaction.Draw` zeichnen. Eine solche `Draw`-Interaktion benötigt beim initialisieren eine Vektor-`source` und einen Geometrietyp.

4.3.1 Vektorlayer und `Draw`-Interaktion erzeugen

Übungen

1. Wir beginnen mit dem unten aufgeführten Beispiel. Öffnen Sie die Datei `map.html` im Texteditor und stellen Sie sicher, dass der Inhalt etwa wie folgt aussieht.

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="ol3/ol.css" type="text/css">
    <style>
    #map {
      height: 256px;
      width: 512px;
    }
    </style>
    <script src="ol3/ol.js" type="text/javascript"></script>
    <title>OpenLayers 3 example</title>
  </head>
  <body>
    <h1>My Map</h1>
    <div id="map"></div>
    <script type="text/javascript">
      var source = new ol.source.GeoJSON({
        url: 'data/layers/7day-M2.5.json'
      });
      var draw = new ol.interaction.Draw({
        source: source,
        type: 'Point'
      });
      var map = new ol.Map({
        interactions: ol.interaction.defaults().extend([draw]),
        target: 'map',
        renderer: 'canvas',
        layers: [
          new ol.layer.Tile({
            title: "Global Imagery",
```

```
            source: new ol.source.TileWMS({
              url: 'http://maps.opengeo.org/geowebcache/service/wms',
              params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
            })
          }),
          new ol.layer.Vector({
            title: 'Earthquakes',
            source: source,
            style: new ol.style.Style({
              image: new ol.style.Circle({
                radius: 5,
                fill: new ol.style.Fill({
                  color: '#0000FF'
                }),
                stroke: new ol.style.Stroke({
                  color: '#000000'
                })
              })
            })
          })
        ],
        view: new ol.View2D({
          projection: 'EPSG:4326',
          center: [0, 0],
          zoom: 1
        })
      });
    </script>
  </body>
</html>
```

2. Speichern Sie Ihre Änderungen und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html

   Klicken Sie mit der linken Maustaste auf die Karte, um neue Features hinzuzufügen.

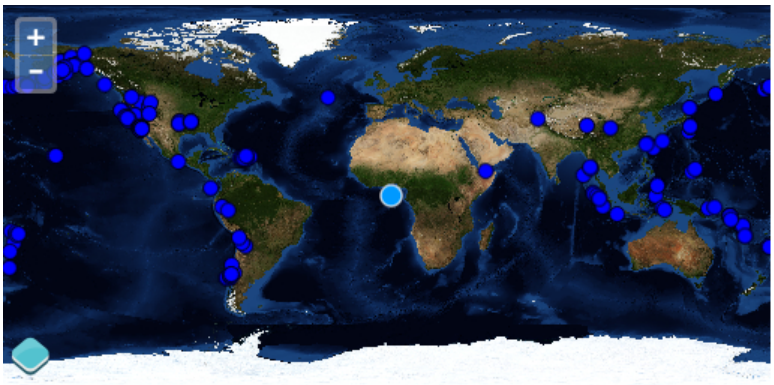


Abbildung 4.3: Featureerzeugung mittels einer `Draw`-Interaktion.

Zusatzaufgabe

1. Erzeugen Sie einen *Eventhandler* der jeweils für ein neu gezeichnetes Feature dessen Koordinaten auf der Konsole ausgibt. (Tipp: der Event lautet *drawend*).

4.4 Features modifizieren

Um Features zu verändern, werden wir eine `ol.interaction.Select` mit einer `ol.interaction.Modify` kombinieren. Diese Interaktionen teilen sich einen gemeinsamen `ol.FeatureOverlay`.

4.4.1 Vektorlayer und `Modify`-Interaktion erzeugen

Übungen

1. Wir beginnen wieder mit dem unten aufgeführten Beispiel. Öffnen Sie die Datei `map.html` im Texteditor und stellen Sie sicher, dass der Inhalt etwa wie folgt aussieht.

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="ol3/ol.css" type="text/css">
    <style>
    #map {
      height: 256px;
      width: 512px;
    }
    </style>
    <script src="ol3/ol.js" type="text/javascript"></script>
    <title>OpenLayers 3 example</title>
  </head>
  <body>
    <h1>My Map</h1>
    <div id="map"></div>
    <script type="text/javascript">
      var source = new ol.source.GeoJSON({
        url: 'data/layers/7day-M2.5.json'
      });
      var overlay = new ol.FeatureOverlay({
        style: new ol.style.Style({
          image: new ol.style.Circle({
            radius: 7,
              fill: new ol.style.Fill({
              color: [0, 153, 255, 1]
            }),
            stroke: new ol.style.Stroke({
              color: [255, 255, 255, 0.75],
              width: 1.5
            })
          }),
          zIndex: 100000
        })
      });
      var modify = new ol.interaction.Modify({ featureOverlay: overlay });
      var select = new ol.interaction.Select({ featureOverlay: overlay });
      var map = new ol.Map({
        interactions: ol.interaction.defaults().extend([select, modify]),
        target: 'map',
        renderer: 'canvas',
        layers: [
          new ol.layer.Tile({
            title: "Global Imagery",
            source: new ol.source.TileWMS({
              url: 'http://maps.opengeo.org/geowebcache/service/wms',
              params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}
            })
```

```
            }),
            new ol.layer.Vector({
              title: 'Earthquakes',
              source: source,
              style: new ol.style.Style({
                image: new ol.style.Circle({
                  radius: 5,
                  fill: new ol.style.Fill({
                    color: '#0000FF'
                  }),
                  stroke: new ol.style.Stroke({
                    color: '#000000'
                  })
                })
              })
            })
          ],
          view: new ol.View2D({
            projection: 'EPSG:4326',
            center: [0, 0],
            zoom: 1
          })
        });
      </script>
    </body>
  </html>
```

2. Speichern Sie Ihre Änderungen und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html

   Klicken Sie mit der linken Maustaste auf die Karte, um ein Erdbeben auszuwählen (`interaction.Select`). Ziehen Sie das Feature anschließend mit der Maustaste an eine neue Lokation (`interaction.Modify`).

4.4.2 Einige Details

Schauen wir uns genauer an, wie wir Features editieren können.

```
var overlay = new ol.FeatureOverlay({
  style: new ol.style.Style({
    image: new ol.style.Circle({
      radius: 7,
      fill: new ol.style.Fill({
        color: [0, 153, 255, 1]
      }),
      stroke: new ol.style.Stroke({
        color: [255, 255, 255, 0.75],
        width: 1.5
      })
    }),
    zIndex: 100000
  })
});
var modify = new ol.interaction.Modify({ featureOverlay: overlay });
var select = new ol.interaction.Select({ featureOverlay: overlay });
```

Wir erzeugen zwei Interaktionen: eine Instanz von `ol.interaction.Select` um Features vor dem editieren auszuwählen, und eine Instanz von `ol.interaction.Modify` um die Geometrien tatsächlich zu verändern. Beiden Interaktionen weisen wir die gleiche Instanz der Klasse `ol.FeatureOverlay` (mit spezifischen Stilangaben, die während Selektion und Modifikation wirksam sind) zu.

Klickt man erneut, so wird das zunächst gewählte / editierte Feature wieder im Stil des Layers gezeichnet.

Styling von Vektorlayern

5.1 Mit Vektorlayern arbeiten

Mit `ol.layer.Vector` steht in ol3 ein durchaus recht komplexer Layertyp zur Verfügung. Standardmäßig werden beim Verwenden dieses Layers von ol3 keine Annahmen dazu gemacht, woher dieser Layer seine Daten bezieht, hierum kümmert sich die Klasse `ol.source.Vector`. Wie man Vektorlayer ausgestaltet, beschreibt *einer der nächsten Abschnitte*. Hier werden wir uns eine zunächst einge Grundlagen zu Vektordaten erarbeiten.

5.1.1 `ol.format`

In ol3 kümmern sich die `ol.format`-Klassen um das Parsen der von einem Server gelieferten Daten, die Features enthalten. Meist werden Sie jedoch nicht die `ol.format`-Klassen **direkt** verwenden, sondern eher spezialisierte `source`-Klassen (wie etwa `ol.source.KML`). Formate transformieren rohe Daten in echte `ol.Feature`-Objekte.

Vergleichen Sie bitte die beiden Datenblöcke weiter unten. Beide repräsentieren das gleiche `ol.Feature`-Objekt (Es handelt sich um einen Punkt in Barcelona). Der erste Block stellt dieses Feature mittels GeoJSON dar (vgl. `ol.format.GeoJSON`). Der zweite Block dagegen enthält KML (OGC Keyhole Markup Language) (vgl. `ol.format.KML`)

GeoJSON-Beispiel

```
{
    "type": "Feature",
    "id": "OpenLayers.Feature.Vector_107",
    "properties": {},
    "geometry": {
        "type": "Point",
        "coordinates": [-104.98, 39.76]
    }
}
```

KML-Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Placemark>
    <Point>
      <coordinates>-104.98,39.76</coordinates>
    </Point>
  </Placemark>
</kml>
```

5.2 Styling Grundlagen

Wenn man HTML-Elemente ausgestaltet, kann man zum Beispiel ein CSS wie das folgende verwenden:

```
.someClass {
    background-color: blue;
    border-width: 1px;
    border-color: olive;
}
```

`.someClass` ist ein Selektor (in diesem Fall einer, der jedwedes Element mit der Klasse `"someClass"` selektieren würde) und der mit geschwungen Klammern umschlossene Block stellt eine Gruppe von Schlüssel-Wert-Paaren dar: die eigentlichen Stildeklarationen.

5.2.1 `ol.feature.StyleFunction`

Ein Vektorlayer akzeptiert als Wert für die `style`-Konfigurationsoption eine Funktion, in welcher ein unterschiedlicher Stil anhand eines Featureattributes zurückgegeben werden kann. Der Funktion werden zwei Argumente übergeben: Das zu stylende `feature` und die aktuelle `resolution`. Nehmen wir an, Sie wollten alle Features, die ein Attribut `class` mit Wert `someClass` besitzen, speziell stylen, so könnte Ihre Stylingfunktion wie folgt aussehen:

```
style: function(feature, resolution) {
  if (feature.get('class') === 'someClass') {
    // return the style
  }
}
```

5.2.2 Stildeklarationsböcke: Symbolizer

Das Äquivalent zu einem Block von Stildeklarationen aus CSS sind in ol3 sogenannte *symbolizer*. Üblicherweise handelt es sich hierbei um Instanzen der Klassen im Namensraum `ol.style`. Um Polygon-Features etwa mit blauem Hintergrund und einem 1-Pixel breiten Rahmen in olivgrün zu zeichnen, würde man zwei *symbolizer* verwenden können:

```
new ol.style.Style({
  fill: new ol.style.Fill({
    color: 'blue'
  }),
  stroke: new ol.style.Stroke({
    color: 'olive',
    width: 1
  })
});
```

Je nach Geometrietyp können verschiedene *symbolizer* verwendet werden. Linien verhalten sich weitestgehend wie Polygone, akzeptieren jedoch keine Füllung. Punkte können derzeit mittels `ol.style.Circle` (für Kreissymbole) oder `ol.style.Icon` (z.B. für png-Bilder) ausgestaltet werden. Schauen wir uns ein Beispiel mit Kreissymbolen an:

```
new ol.style.Circle({
  radius: 20,
  fill: new ol.style.Fill({
    color: '#ff9900',
    opacity: 0.6
  }),
  stroke: new ol.style.Stroke({
    color: '#ffcc00',
    opacity: 0.4
  })
});
```

5.2.3 ol.style.Style

Jedes ol.style.Style-Objekt hat vier Schlüssel: fill, image, stroke und text. Optional existiert eine zIndex-Eigenschaft. Die Stylefunktion gibt ein Array von ol.style.Style-Objekten zurück.

Angenommen alle Features sollten rot gezeichnet werden, außer denjenigen, die ein class-Attribut mit dem Wert "someClass" haben (und diese sollen wie oben blau gefüllt sein und einen 1-Pixel breiten Rand in oliv haben), so könnte die Funktion wie folgt aussehen.

```
style: (function() {
  var someStyle = [new ol.style.Style({
    fill: new ol.style.Fill({
      color: 'blue'
    }),
    stroke: new ol.style.Stroke({
      color: 'olive',
      width: 1
    })
  })];
  var otherStyle = [new ol.style.Style({
    fill: new ol.style.Fill({
      color: 'red'
    })
  })];
  return function(feature, resolution) {
    if (feature.get('class') === "someClass") {
      return someStyle;
    } else {
      return otherStyle;
    }
  };
}())
```

Wenn möglich, sollten die tatsächlichen Stil-Objekte außerhalb der Funktion möglichst nur einmal erzeugt werden, und in der Funktion nur Referenzen hierauf zurückgegeben werden (Bessere Perfomance). Im obigen Beispiel wird hierzu eine *closure* verwendet.

Bemerkung: Auch Features akzeptieren in ihrer style-Konfigurationsoption eine Funktion. Jene wird mit der aktuellen resolution aufgerufen und erlaubt ein sehr indivuelles Stylen je Feature und Maßstab.

5.2.4 Pseudoklassen

CSS kennt das Konzept sogenannter Pseudoklassen, die die Anwendung von Stildeklaration weiter einschränkt um spezielle Kontexte abzubilden, die nicht oder nur schwerlich über *klassische* Selektoren abgebildet werden können (z.B. *:hover* oder *:active*).

In ol3 ist ein in mancher Hinsicht ähnliches Konzept mittels der style-Option von ol.FeatureOverlay umgesetzt. Um selektierte Features einer ol.interaction.Select zu visualisieren, wird ein solcher *feature overlay* verwendet, dessen style das Aussehen bestimmt.

Ein Beispiel wäre etwa:

```
var select = new ol.interaction.Select({
  featureOverlay: new ol.FeatureOverlay({
    style: new ol.style.Style({
      fill: new ol.style.Fill({
        color: 'rgba(255,255,255,0.5)'
      })
    })
  })
});
```

Nachdem wir die Basiskonzepte nun kennen, können wir uns nun der konkreten *Ausgestaltung von Vektorlayern* zuwenden.

5.3 Vektorlayer ausgestalten

Übungen

1. Wir beginnen mit einem lauffähigen Beispiel, welches Gebäudegrundrisse in einem Vektorlayer darstellt. Öffnen Sie die Datei `map.html` im Texteditor und fügen Sie den nachfolgenden Code ein:

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="ol3/ol.css" type="text/css">
    <style>
    #map {
      background-color: gray;
      height: 256px;
      width: 512px;
    }
    </style>
    <title>OpenLayers 3 example</title>
    <script src="ol3/ol.js" type="text/javascript"></script>
  </head>
  <body>
    <h1>My Map</h1>
    <div id="map"></div>
    <script type="text/javascript">
      var map = new ol.Map({
        target: 'map',
        renderer: 'canvas',
        layers: [
          new ol.layer.Vector({
            title: 'Buildings',
            source: new ol.source.KML({
              url: 'data/layers/buildings.kml'
            }),
            style: new ol.style.Style({
              stroke: new ol.style.Stroke({color: 'red', width: 2})
            })
          })
        ],
        view: new ol.View2D({
          projection: 'EPSG:4326',
          center: [-122.791859392, 42.3099154789],
          zoom: 16
        })
      });
    </script>
  </body>
</html>
```

2. Speichern Sie Ihre Änderungen und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html

   Sie sollten rot umrandete Gebäudegrundrisse sehen.

3. Nachdem uns *Styling in ol3*, nicht mehr fremd ist, können wir eine Stylefunktion erzeugen, die je nach der Größe des Grundrisses die Features einfärbt. Ersetzen Sie die `style`-Konfigurationsoption des Layers mit folgenden Zeilen:

```
style: (function() {
  var defaultStyle = [new ol.style.Style({
    fill: new ol.style.Fill({color: 'navy'}),
    stroke: ol.style.Stroke({color: 'black', width: 1})
  })];
  var ruleStyle = [new ol.style.Style({
    fill: new ol.style.Fill({color: 'olive'}),
    stroke: new ol.style.Stroke({color: 'black', width: 1})
  })];
  return function(feature, resolution) {
    if (feature.get('shape_area') < 3000) {
      return ruleStyle;
    } else {
      return defaultStyle;
    }
  };
})()
```

4. Speichern Sie Ihre Änderungen und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html

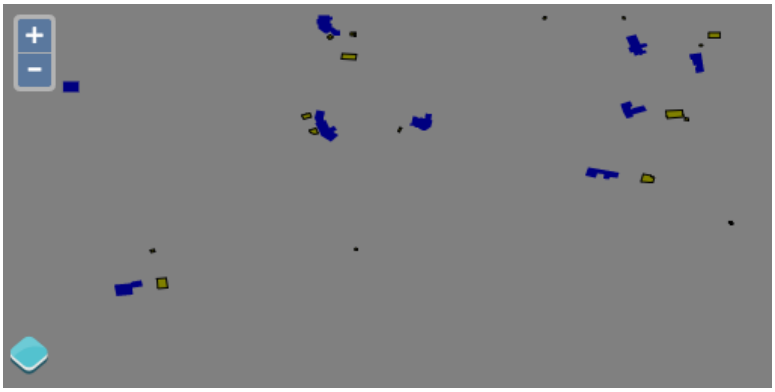


Abbildung 5.1: Die Grundrisse werden nun nach Grundfläche ausgestaltet.

5. Schließlich wollen wir unsere Features nun noch mit einem textlichen *Label* versehen:

```
style: (function() {
  var stroke = new ol.style.Stroke({
    color: 'black'
  });
  var textStroke = new ol.style.Stroke({
    color: '#fff',
    width: 3
  });
  var textFill = new ol.style.Fill({
    color: '#000'
  });
  return function(feature, resolution) {
    return [new ol.style.Style({
      stroke: stroke,
      text: new ol.style.Text({
        font: '12px Calibri,sans-serif',
        text: feature.get('key'),
        fill: textFill,
        stroke: textStroke
      })
    })];
  };
})()
```

6. Speichern Sie Ihre Änderungen und laden Sie die Seite im Browser neu: http://localhost/ol3-ws/ol3-training-master/map.html

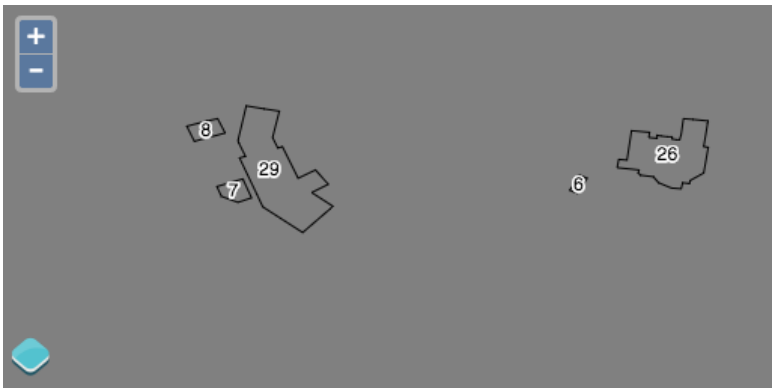


Abbildung 5.2: Die Gebäude werden nun auch mit einem *Label* gerendert.