OpenLayers 3 – Einführung, Verwendungsbeispiele und technische Highlights

Marc Jansen & Andreas Hocevar

OpenLayers ist eine OpenSource JavaScript Kartenbibliothek mit sehr großer Verbreitung, sowohl innerhalb von OSGeo-Projekten als auch in privaten wie öffentlichen Webseiten und Internet-/Intranet-Applikationen. Die 2.x-er Versionen der Software werden bis zum heutigen Tage weiterentwickelt und gepflegt. Doch natürlich nagt der Zahn der Zeit auch an OpenLayers: Entwickler und Anwender haben 2014 verständlicherweise andere Ansprüche an digitale Kartenbibliotheken, als dies vor 8 Jahren der Fall war. Vor allem aber haben sich Web-Browser rasant weiterentwickelt, so dass viele Programmierumwege aus der Frühzeit von OpenLayers 2 heute nicht mehr notwendig sind.

Bereits seit einiger Zeit wird daher von der Entwicklergemeinde an OpenLayers 3 gearbeitet, zum Zeitpunkt der Einreichung dieses Artikel ist die aktuellste Version 3.0.0.beta.2.

Was ist neu bei OpenLayers 3?

OpenLayers 3 (ol3, [1]) wurde von Grund auf neu entwickelt und daher ist de facto alles neu an der Bibliothek.

Das neue ol3 nutzt intern die JavaScript-Bibliothek Google Closure [2], die auch in zahlreichen Google Produkten (etwa Gmail oder Google Maps) verwendet wird. Closure wird in ol3 vor allem verwendet, um das Klassensystem bereitzustellen und wiederkehrende Aufgaben (wie etwa HTML-Elementerzeugung, Eventhandling und DOM-Manipulation) zu lösen. Um die stärkestmögliche Kompression des JavaScript-Codes zu erzielen, kommt der Closure Compiler [3] zum Einsatz. Insbesondere der letzte Punkt garantiert eine sehr kleine Dateigröße der zusammengefassten und komprimierten ol3-JavaScript-Datei. Applikationsentwickler/ können die ol3-Bibliothek natürlich auch ohne Google Closure zu benutzen. Bereits in der Entwicklung wurde Wert darauf gelegt, dass ol3 unproblematisch auch mit anderen JavaScript-Frameworks oder Bibliotheken einzusetzen ist.

Die sicherlich bemerkenswerteste Neuerung von ol3 ist die zusätzliche Unterstützung von WebGL als Rendering Engine. WebGL [4] erlaubt es, in modernen Browsern hardwarebeschleunigte 3D-Grafiken darzustellen und dies ohne zusätzliches Plugin. Unterstützung für WebGL ist in den aktuellen Versionen der Browser Google Chrome, Chromium, Opera, Firefox und Internet Explorer bereits nativ integriert. Mit WebGL-Unterstützung in ol3 ist es nunmehr möglich, mit **einer** Bibliothek sowohl hochperformante 2D Karten im Web darzustellen, als auch 3D-Ansichten zu erstellen, die im Browser bedient werden. Für Darstellungen in virtuellen Globen wird auf die Integration der Cesium Bibliothek [5] gesetzt. Mit der aktuellen Version von ol3 (beta.2) ist noch keine 3D-Funktionalität implementiert, die grundsätzliche Funktionalität und Architektur ist jedoch in Form einer Vektor-Renderer Abstraktionsschicht bereits verfügbar. Im Zuge der Einführung dieser Abstraktionsschicht hat sich das ol3-API seit der beta.1 Version auch stark verändert.

OpenLayers 2 wurde lange vor dem Boom des mobilen Internets, auf welches via Smartphones und Tablets zugegriffen wird, entwickelt. Die Unterstützung mobiler Endgeräte wurde zwar nachgerüstet und ist funktional, doch ol3 unterstützt mobile Endgeräte von Anfang an. Hierbei wurden in der Entwicklung stets die besonderen Limitierungen (langsame/instabile Netzverbindung, kleiner Arbeitsspeicher, vergleichsweise kleiner Bildschirm etc.) und zusätz-

lichen Möglichkeiten (MultiTouch-Bedienung, Rotation des Gerätes, HTML5-APIs etwa zur Geolokalisierung, etc.) mobiler Geräte beachtet.

Zwei größere Kritikpunkte an OpenLayers 2 waren die z.T. fragmentierte und unvollständige Dokumentation und die Schwierigkeit, eine speziell optimierte Version von OpenLayers zu erzeugen, die nur enthält, was man wirklich benötigt (sogenannter custom build). Hier möchte ol3 besser sein: Bereits heute ist es möglich, die Bibliothek so zu nutzen, dass ein minimaler build einfach zu erzeugen ist. Geplant sind hier weitere Tools und/oder Applikationen, die diese Anpassungen noch einfacher für Endanwender und Programmierer machen.

Bei der Dokumentation der öffentlichen Schnittstelle setzt ol3 derzeit auf das Programm jsdoc3 [6], welches bereits hervorragende, komplette und konsistente Dokumentation produziert (vgl. [7]). Hier sind sicherlich noch weitere Anpassungen und Ergänzungen seitens der ol3-Entwickler vorzunehmen, aber wir sind sicher, die Qualität und Quantität der Dokumentation gegenüber OpenLayers 2 deutlich zu erhöhen.

Was bleibt gleich?

Auch wenn der Code neu geschrieben wurde, so sind doch die Ziele hinter OpenLayers unverändert. Auch ol3 ist:

- Vielseitig verwendbar
- Modern implementiert
- Performant programmiert
- Erweiter- und anpassbar
- Einfach zu verstehen
- Standardkonform
- Cross-Anything (Browser, Plattform & Endgerät)
- Vollständig dokumentiert
- Verlässlich in der Verwendung

Die Bibliothek ol3 erlaubt zahlreiche Verwendungsmöglichkeiten und stellt dem Entwickler die Werkzeuge bereit, um so unterschiedliche Aufgaben wie einen einfachen Kartenviewer für eine Homepage zu erzeugen aber auch eine komplette webbasierte GDI (Geodateninfrastruktur) mit optionaler 3D Visualisierung umzusetzen.

Moderne Webtechnologien sind auch in OpenLayers 2.x verwendbar gewesen. ol3 macht eben dort weiter und unterstützt zahlreiche HTML5-APIs, das bereits erwähnte WebGL und nutzt die Möglichkeiten von CSS3 wo möglich und sinnvoll.

Performance ist in ol3 von hoher Bedeutung. Dies betrifft im wesentlichen zwei Aspekte: Einerseits sollte die Bibliothek die kleinstmögliche Dateigröße haben (vgl. Verwendung von Closure Compiler, s.o.) und andererseits sollten interne Methoden so performant wie möglich programmiert sein. Beide Aspekte haben in der Entwicklung von ol3 einen hohen Stellenwert.

Eine Bibliothek kann nie alles für jeden Entwickler und Anwendungsfall bereitstellen. Auch ol3 möchte es den Benutzern von ol3 einfach machen, das Verhalten von ol3 zu erweitern oder zu beeinflussen. Via Events kann der Entwickler bereits jetzt an entscheidenden Stellen auf Darstellung und Verhalten der Bestandteile von ol3 einwirken. Alle visuellen Elemente von OpenLayers lassen sich einfach mittels CSS an eigene Bedürfnisse anpassen. Im Gegensatz zu OpenLayers 2 bringt ol3 weniger allgemeine (d.h. nicht geobezogene) Funktiona-

lität mit, lässt sich aber sehr einfach mit anderen JavaScript-Bibliotheken und Frameworks (wie z.B. jQuery [8], o.ä.) kombinieren, welche diese Funktionalität bereitstellen.

Hauptaugenmerk von OpenLayers 2 war und ist es, einfach bedien- und verwendbar zu sein. Auch ol3 legt hier einen Schwerpunkt. Die API von ol3 ist so konzipiert, dass Sie vom Anwender einfach erfasst werden kann und keine inhaltlichen Brüche zeigt. Verwendet man erfolgreich Komponente A in einer bestimmten Weise, kann man sich darauf verlassen, dass eine ähnliche Komponente B üblicherweise nahtlos anstelle von A verwendet werden kann. Hier ist natürlich auch das frühe Feedback der Anwender hoch geschätzt. Noch ist die API an vielen Stellen im Fluss, daher ist die Diskussion um jene gerne gesehen.

ol3 wird die bekanntesten und weit verbreiteten Standards (etwa des OGCs) wie WMS, WFS etc. natürlich unterstützen. Zusammen mit den Zielen des Cross-Everything (Funktionalität ist browser-, plattform- und Endgeräteunabhängig gewährleistet) kann man mit ol3 zukunftsfähige Applikationen entwickeln.

Durch eine vollständige Dokumentation der API und ergänzende Tutorials sowie eine breit getestete Funktionalität ist ol3 bereits heute vor allem eines: Verlässlich. OpenLayers 2.x hat in den letzten acht Jahren bei insgesamt 13 neuen Releases stets die bekannte API nicht gebrochen, so dass ein Upgrade der OpenLayers-Version meist trivial war. Hier möchte ol3 aufbauen und das Vertrauen in die Updatefähigkeit nicht enttäuschen.

Verwendungsbeispiele

Was kann ol3 nun schon zum jetzigen Zeitpunkt? Im Folgenden werden wir drei aktuelle offizielle Beispiele kurz vorstellen und auch auf zwei Realweltanwendungen, die ol3 verwenden, verweisen.

Das Beispiel "Simple example" [9] zeigt, wie die Hauptkomponenten **`ol.Map`**, **`ol.layer.Layer`** (hier ein **`Tile`**-Layer) und **`ol.View2D`** zu verwenden sind:

```
var map = new ol.Map({
  layers: [
    new ol.layer.Tile({
      source: new ol.source.OSM()
    })
  ],
  target: 'map',
  view: new ol.View2D({
    center: [0, 0],
    zoom: 2
  })
});
```

Die **`ol.Map`** als Kernkomponente von ol3 benötigt i.d.R. wenigstens ein Kartenthema, eine View-Definition (im Falle einer 2D-Karte üblicherweise mit Zentrum und Zoomlevel) sowie ein HTML-Element in welchem die Karte gerendert werden soll (**`target: 'map'`**, hier ist **`map`** die **`id`** eines **`<div>`**-Elements auf der Seite).

Mit diesen zwölf Zeilen Code wird auf der Webseite eine voll funktionale Karte gezeichnet, die eine OpenStreetMap-Quelle verwendet (**`ol.source.OSM`**):

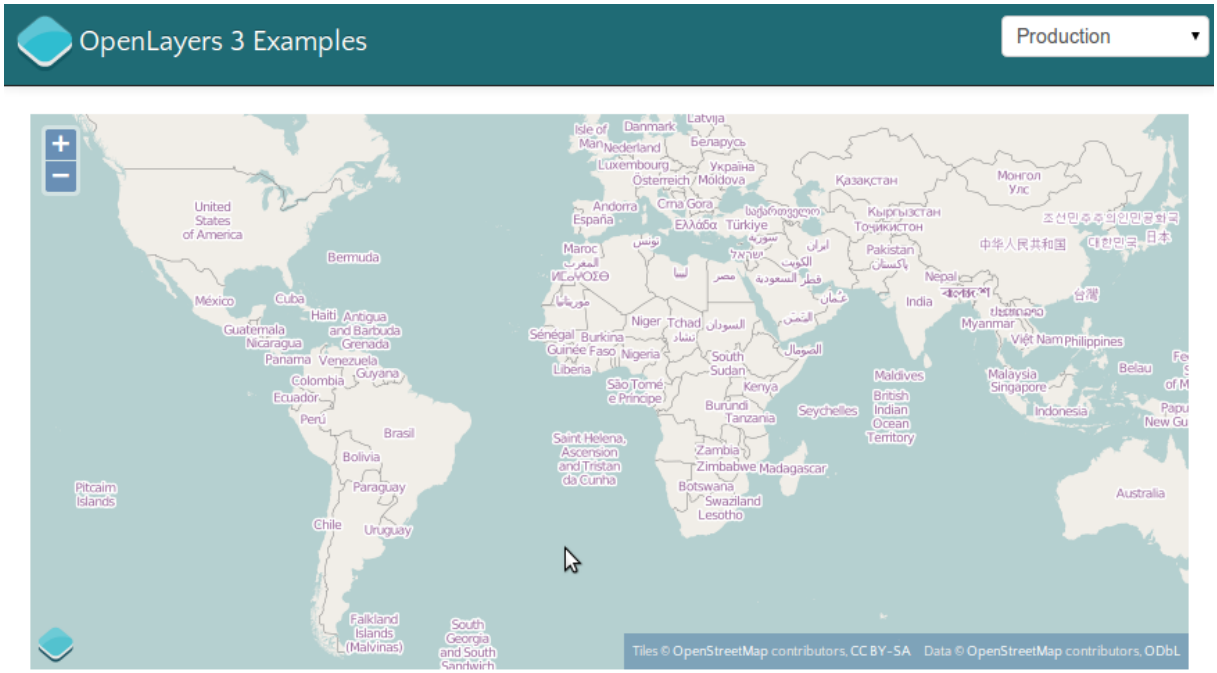

Abbildung 1: Eine einfache ol3-Karte

Am nächsten Beispiel "IGC example" [10], werden wir sehen, dass ol3 auch eine Vielzahl von Vektordaten performant darstellen und mit jenen interagieren kann.

Im Beispiel werden 5 Paragliderflüge im IGC-Format [11] als Quelle (**`ol.source.IGC`**) für einen Vektorlayer verwendet, die insgesamt beinahe 50.000 unterschiedliche Koordinaten enthalten. Beim Überfahren der Karte mit der Maus werden Informationen zum dem Mauszeiger am nächsten befindlichen Flug angezeigt. Diese Interaktion mit der Karte geschieht über einen **`mousemove`**-EventHandler auf einem ol3-Element (im Code unten nicht aufgeführt).

```
// Erzeugung der vectorSource:
var vectorSource = new ol.source.IGC({
  urls: [
    'data/igc/Clement-Latour.igc',
    'data/igc/Damien-de-Baenst.igc' // etc.
  ]
});
// Erzeugung einer Funktion, die das Aussehen der
// Features je Auflösung bestimmt (gekürzt):
var styleFunction = function(feature, resolution) {
  // ...
  return styleArray;
};
// Verwendung der source in ol.layer.Vector:
var layer = new ol.layer.Vector({
  source: vectorSource,
  styleFunction: styleFunction
});
```

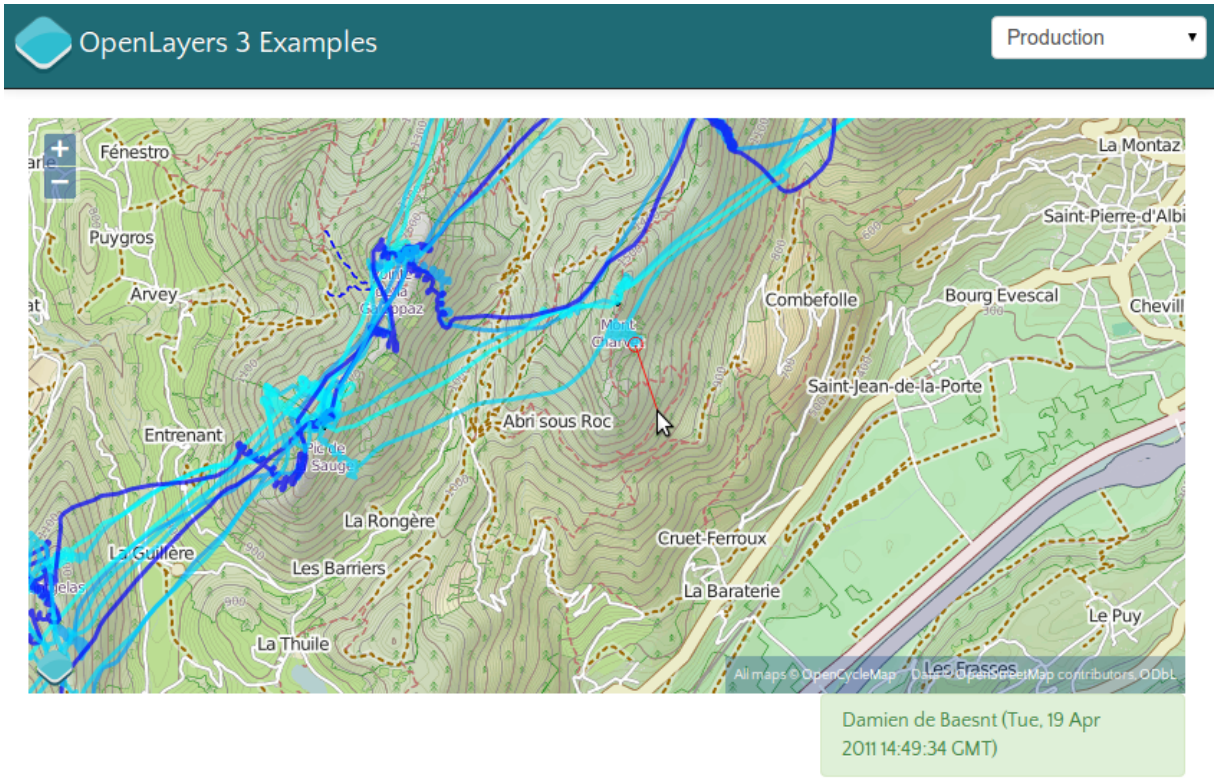

Abbildung 2: Das IGC-Beispiel

Das "Drag & Drop" Beispiel [12] zeigt schließlich, wie man in modernen Browsern mittels Drag & Drop von Vektor-Dateien in verschiedenen Formaten (z.B. GPX, KML, GeoJSON) ein neues Kartenthema hinzufügen kann:

```
  // Erzeugung einer spezifischen Interaktion:
  var dragNDropInteraction = new ol.interaction.DragAndDrop({
    formatConstructors: [
      ol.format.GPX,
      ol.format.GeoJSON // etc.
    ]
  });

// Erweitern der Standardinteraktionen &
  // Erzeugung der Karte mit den Interaktionen
  var interactions = ol.interaction.defaults();
  var map = new ol.Map({
    interactions: interactions.extend([ dragNDropInteraction ]),
    // ...
  }

  // ...wenn Features via drag'n drop hinzugefügt werden...
  dragNDropInteraction.on('addfeatures', function(event) {
    // ...neue Quelle erzeugen...
    var vectorSource = new ol.source.Vector({
      features: event.features,
      projection: event.projection
    });
```

```
    // ...der Karte hinzufügen...
    map.getLayers().push(new ol.layer.Vector({
      source: vectorSource,
      styleFunction: styleFunction
    }));
    var view2D = map.getView().getView2D();
    // ...Rezentrieren des Views.
    view2D.fitExtent(
      vectorSource.getExtent(),
      map.getSize()
    );
  });
```

Dieser Code reicht aus, um einen neuen Vektor-Layer auf die Karte zu bringen, wann immer eine Datei in einem der konfigurierten Formate in das Kartenfenster gezogen wird.

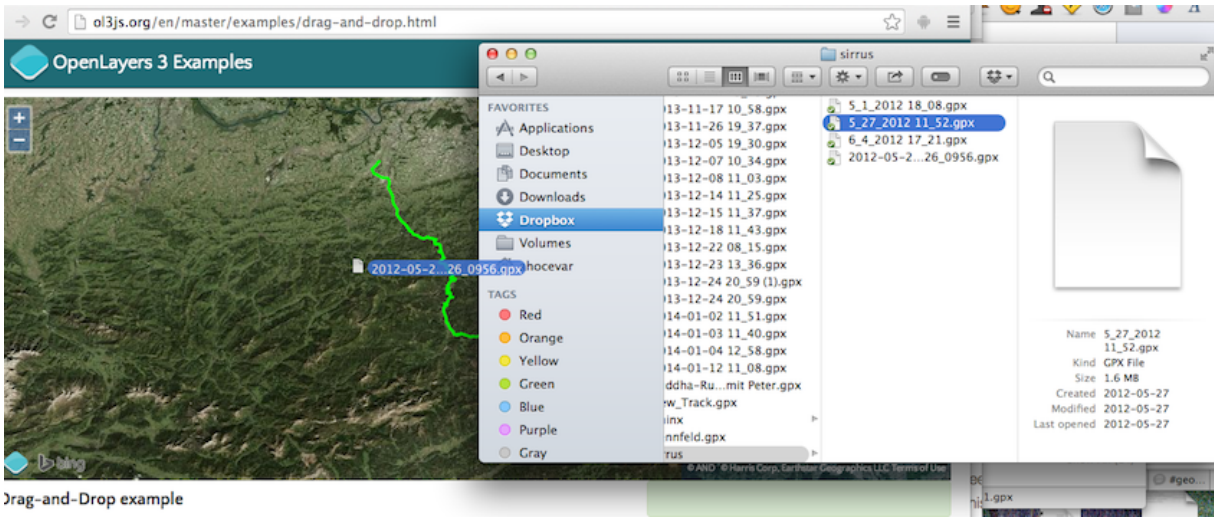

Abbildung 3: Das Drag-and-Drop Beispiel

Auch abseits der offiziellen Beispiel kommt OpenLayers 3 bereits zum Realwelteinsatz:

Im Rahmen der OpenGeo Suite wird für ol3 bereits so genannter Early Access Support gewährt [13]. Daneben ist ol3 die Kartenbibliothek der Wahl im Produkt Mapmeter [14].

Erst kürzlich wurde die u.a. auf ol3 basierende neuste Inkarnation des Kartenportals des Bundesamt für Landestopografie der Schweiz (swisstopo, [15]) veröffentlicht. Der Code hinter der OpenSource Webanwendung ist frei verfügbar [16] und zeigt das effiziente Zusammenspiel von ol3 mit den Frameworks AngularJS [17] und Bootstrap [18].

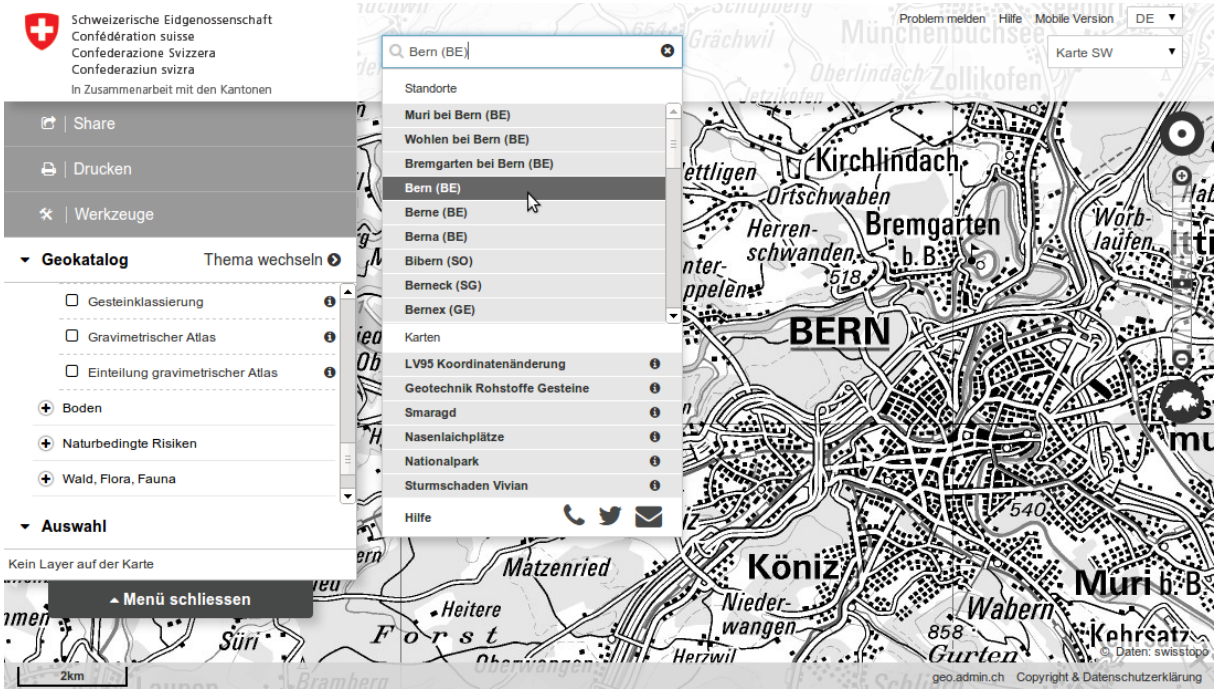

Abbildung 4: Die swisstopo-Anwendung

Technische Highlights

OpenLayers 3 bringt von Grund auf eine 3D-taugliche Architektur. Um dem Rechnung zu tragen, gibt es die **`View`**-Klasse. Derzeit ist nur **`View2D`** implementiert, aber es ist konzeptionell alles für dreidimensionale Darstellungen vorbereitet.

Um sowohl Canvas 2D als auch WebGL für Vektor-Features verwenden zu können, gibt es eine Abstraktion auf der Ebene einfacher Grafikoperationen (Zeichnen von Punkten, Linien, Polygonen, Text). Für Anwendungsentwickler ist es möglich, direkt Zugriff auf den Output-Canvas zuzugreifen, und damit eigene Grafik-Manipulationen vorzunehmen. Intern gibt es ein Replay-API, welches einmal vorberechnete Grafiken mit minimalem Rechenaufwand jederzeit wieder auf den Ziel-Canvas zeichnen kann. Damit ist es möglich, zigtausende Vektor-Features nicht nur ohne merkbare Verzögerung zu rendern, sondern auch zu animieren.

Die Beispiele "Layer spy" [19] und "Layer swipe" [20] zeigen anschaulich, wie mittels **`pre`**- und **`postcompose`** Einfluss auf das letztlich gezeichnete Kartenbild genommen werden kann. Jeweils werden nur Teile verschiedener Layer kombiniert, um ein interaktives und kontextabhängiges Kartenerlebnis zu gestalten.

Mit ol3 ist es einfacher geworden, mit anderen Bibliotheken zusammen eine moderne Webapplikation zu entwickeln. Hier sei beispielhaft auf das "d3 integration"-Beispiel [21] verwiesen. D3 [22] – eine populäre JavaScript-Bibliothek für dynamische, interaktive und vornehmlich grafische Anwendungen – dient hier als Quelle eines **`ol.layer.Image`** und kann damit wie jedes andere Kartenthema verwendet werden.

Kontakt zu den Autoren

Marc Jansen
terrestris GmbH & Co. KG
Pützchens Chaussee 56
53227 Bonn
+49 (228) 962 899 53
jansen@terrestris.de

Andreas Hocevar
c/o Boundless
222 Broadway, 19th Floor
New York, NY 10038, USA
+1 (917) 460-7194
ahocevar@boundlessgeo.com

Literatur & Links

[1] http://ol3js.org/

[2] https://developers.google.com/closure/library/

[3] https://developers.google.com/closure/compiler/

[4] http://www.khronos.org/webgl/

[5] http://cesiumjs.org/

[6] http://usejsdoc.org/

[7] http://ol3js.org/en/master/apidoc/

[8] http://jquery.com/

[9] http://ol3js.org/en/master/examples/simple.html

[10] http://ol3js.org/en/master/examples/igc.html

[11] http://www.ukiws.demon.co.uk/GFAC/documents/tech_spec_gnss.pdf

[12] http://ol3js.org/en/master/examples/drag-and-drop.html

[13] http://boundlessgeo.com/2013/11/blog-opengeo-suite-4-0-released/

[14] http://mapmeter.com/

[15] http://map.geo.admin.ch/

[16] https://github.com/geoadmin/mf-geoadmin3

[17] http://angularjs.org/

[18] http://getbootstrap.com/

[19] http://ol3js.org/en/master/examples/layer-spy.html

[20] http://ol3js.org/en/master/examples/layer-swipe.html

[21] http://ol3js.org/en/master/examples/d3.html

[22] http://d3js.org/