Dissémination de données géoréférencées – qgis-server et openlayers

J.-M Friedt, É. Carry, 14 octobre 2016

Tout possesseur de téléphone portable devient en mesure aujourd'hui de produire de la donnée géoréférencée, qu'il s'agisse de photographies numériques ou de traces de parcours pour les données brutes, ou des données issues d'informations géoréférencées acquises. Nous avons par exemple présenté au FOSS4G-fr [1] et dans ces pages [2] la génération de modèles de batiments par les photographies géoréférencées prises par téléphone, et leur intégration dans QGis sur fond de carte OpenStreetMaps (OSM).

Obtenir de telles données est bien, mais les partager avec une communauté d'utilisateurs, c'est mieux. Le mode de transmission le plus pratique pour les volumes de données dont il est question est évidemment internet, ne serait-ce que pour centraliser les données partagées en vue de garantir leur cohérence si plusieurs utilisateurs les manipulent.

Trois protocoles de dissémination ont été mis en place à cet effet [3, 4] : WM(T)S, WFS et WCS. Le premier dissémine des images représentant les données traitées, avec une résolution adaptée à l'échelle de la carte de l'utilisateur. Ne fournissant qu'une image des informations transmises, il n'est plus possible pour l'utilisateur de s'approprier ou re-traiter lui-même des informations. Les deux autres protocoles disséminent les données brutes, vectorielles ou matricielles. Nous nous proposons d'aborder séquentiellement trois problèmes : un serveur de données pour disséminer les informations qui auront été traitées dans QGis (`qgis-server`) ; la récupération des informations sur un logiciel dédié tel que Qgis pour le traitement des informations ; et finalement la récupération des données dans un client web pour affichage comme couche Openlayers. Afin de ne pas faire durer le suspens plus longtemps, le lecteur est encouragé à consulter `qgis.sequanux.org/jmfwms.html` et `jmfriedt.sequanux.org/reproj.html` pour se faire une idée du résultat recherché.

Le contexte de notre étude porte sur le retrait de glaciers à front marin en milieu arctique. Les images des satellites Landsat sont acquises depuis les années 1970 et mises à disposition par l'USGS, par exemple à `http://landsatlook.usgs.gov/viewer.html`. Elles sont plus ou moins bien géoréférencées (une petite correction est parfois nécessaire, surtout pour les plus anciennes), mais surtout souffrent d'une résolution médiocre (de l'ordre de 30 m/pixel) compte tenu des standards actuels. Plus de 40 ans d'histoire d'images satellites en font néanmoins une source irremplaçable pour appréhender l'évolution des régions qui vont nous intéresser. Notre travail initial sous QGis a été d'importer toutes ces images, les reprojeter dans un référentiel localement plan (WGS84/UTM33N, le référentiel projeté approprié pour le nord de la Norvège tel que nous en informe `http://spatialreference.org/ref/epsg/wgs-84-utm-zone-33n/`), s'assurer par quelques points de référence que les images sont convenablement positionnées, et dans le cas contraire les repositionner au moyen de 4 ou 5 points de référence sur le pourtour de la région considérée (presqu'île de Brøgger, Svalbard). Ces informations matricielles sont alors utilisées pour tracer manuellement (création d'un *shapefile* comprenant des lignes) les fronts marins des glaciers, créant ainsi un jeu de données vectorielles. Par ailleurs, un trajet en avion a été enregistré au moyen du récepteur GPS d'un téléphone portable (logiciel `OSMTracker` sous Android) et nous désirons insérer la séquence de points au format GPX dans nos cartes. Sous QGis, l'opération est triviale (Vector → GPS Tools → Load GPX File) mais pourrons nous exporter cette information par le web et l'inclure dans les pages affichées par un navigateur ? Nous verrons que la réponse n'est pas aussi triviale qu'il pourrait paraître.

1 Le serveur : `qgis-server`

QGis a tout prévu pour nous : un projet QGis s'exporte dans un format accessible par le web au travers de `qgis-server`. Le principal exercice va donc porter sur l'installation de cet outil, puisque exporter le projet se résume à Project → Projet Properties et activer WMS, WFS et WCS en cochant toutes les couches (Select All) dans les deux derniers cas.

L'installation de `qgis-server` ne présente aucune difficulté et est décrite parfaitement à `http://docs.qgis.org/testing/en/docs/user_manual/working_with_ogc/ogc_server_support.html`. On notera cependant que bien que le service WM(T)S fonctionne convenablement avec des versions plus anciennes de `qgis-server`, l'export des couches vectoriel (formats WFS et WCS) ne semble pas opérationnel pour les version précédent 2.12. Nous expérimenterons donc avec une version au moins supérieure à 2.14

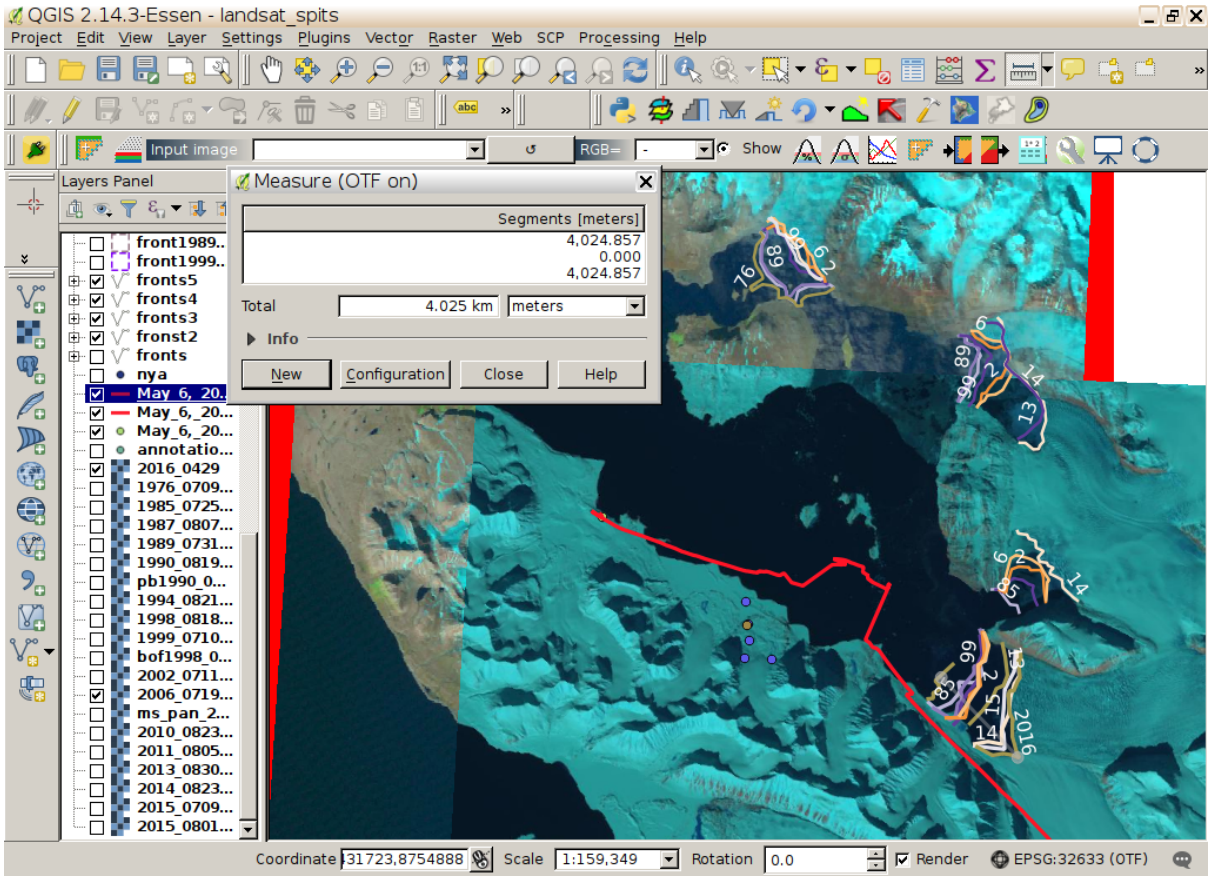

FIGURE 1 – Projet QGis que nous désirons partager au travers d'internet, avec toutes les images sans nuages acquises par Landsat de la presqu'île de Brøgger au Svalbard, et la mesure de position des fronts, illustrant le recul de 4 km en 40 ans du Kongsbreen.

de QGis et de son serveur `qgis-server`. Sous Debian GNU/Linux, cela signifie éviter wheezy mais travailler dans la version stable à la date de cette rédaction.

À l'issue de l'installation du serveur comme service d'apache2 [1], le bon fonctionnement du service est validé (dans le cas d'une installation locale) par

```
wget -O - "http://127.0.0.1/cgi-bin/qgis_mapserv.fcgi?SERVICE=WFS&VERSION=1.0.0&REQUEST=GetCapabilities"
```

qui doit se traduire par une réponse du type

```
<Service>
  <Name>WMS</Name>
  <!-- Human-readable title for pick lists -->
  <Title>QGIS mapserver</Title>
  <!-- Narrative description providing additional information -->
  <Abstract>A WMS service with QGIS mapserver</Abstract>
...
```

Le service étant fonctionnel, divers projets QGis (fichier `.qgs`) sont placés, avec toutes les données associées aux diverses couches du projet, dans des sous-répertoires de `/usr/lib/cgi-bin`, en complétant avec une copie du serveur Fast-CGI (`qgis_mapserv.fcgi`) [2]. Si le projet QGis autorise l'export des couches au format WM(T)S ou WFS, nous validons la capacité à rapatrier ces couches (dans notre cas le projet et ses fichiers se trouvent dans `/usr/lib/cgi-bin/project/landsat` de `qgis.sequanux.org`) par

1. nécessitant le support FCGI fourni par `libapache2-mod-fcgid`
2. attention, les liens symboliques entre `/usr/lib/cgi-bin` et, par exemple, un répertoire utilisateur, ne sont autorisés qu'en modifiant le configuration par défaut de Apache2 sous debian dans `/etc/apache2/conf-available/serve-cgi-bin.conf` en remplaçant le "+" par un "-" de l'option `SymLinksIfOwnerMatch`.

```
$ wget -O - "http://qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi?SERVICE=WFS&VERSION=1.0.0&\
REQUEST=GetCapabilities"
```

qui répond par la liste des couches accessibles sous la forme

```
<WFS_Capabilities xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ...
 <Service>
  <Name>WFS</Name>
...
  <FeatureType>
   <Name>fronts5</Name>
   <Title>fronts5</Title>
   <Abstract></Abstract>
   <SRS>EPSG:4326</SRS>
   <Operations>
    <Query/>
   </Operations>
   <LatLongBoundingBox maxx="12.2424" minx="12.103" maxy="79.0268" miny="78.9989"/>
  </FeatureType>
...
```

Nous voyons donc que la projection dans laquelle la coordonnées des points formant la couche (ici vectorielle) est renseignée (EPSG :4326 signifie WGS84 en coordonnées sphériques tel que nous l'indique le CRS sous QGis), et l'extension de la couche (autour de 12°E et 79°N).

Ces points sont récupérés – au format GML – par une requêtre WFS de la forme :

```
$ wget -O - "http://qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi?SERVICE=WFS&VERSION=1.0.0&\
REQUEST=GetFeature&typename=fronts5&featureid=fronts5.toto"
```

qui répond

```
   <gml:LineString srsName="EPSG:4326">
    <gml:coordinates cs="," ts=" ">12.11525618,79.02193675 12.1224397,79.02118041 12.11728195,
79.01669523 12.11808149,79.0123295 12.12970628,79.00850923 12.14676275,79.00549989 12.16023071,79.00480137
12.16426326,79.00331772 12.16469826,79.00161054 12.17939472,78.99997196 12.18340417,79.00381158 12.20189733,
79.00556583 12.21455655,79.00542793 12.2268052,79.00560265 12.2366082,79.00626179 12.24175315,79.00561112
</gml:coordinates>
```

Nous avons ici les diverses coordonnées qui forment la ligne d'une des couches vectorielles de notre projet QGis, accessible au travers d'une requête HTML. Ces données sont donc accessibles par la communauté, il reste à en faire un bon usage avec les divers outils à notre disposition. Nous présentons quelques exemples ci-dessous.

2 Le client : Qgis

Le premier environnement le plus simple pour accéder à ces données est QGis lui même. Dans la barre des icônes de gauche de QGis, les trois icônes en forme de globe proposent l'accès respectivement aux données au format WM(T)S, WCS et WFS. Dans tous les cas, il nous faut renseigner l'url du serveur, par exemple `http://qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi`. Lors de la connexion, la liste des couches accessibles est proposée, et cliquer sur une couche suivi de "Add" donne accès au jeu de données. La seule subtilité dans cette application tient à ne pas utiliser un serveur trop ancien (avant 2.12) sous peine de voir la liste des couches s'afficher, mais une liste vide de points être retournée par les requêtes WFS. WM(T)S semble avoir toujours bien fonctionné (Fig. 2).

3 Intégration dans une page web : `openlayers` 2

La portée de notre serveur se retreint encore à un public relativement averti puisque utilisateur de QGis. Nombreuses sont les cartes sur le web qui ne sont pas destinées à un auditoire de géographes mais simplement pour illustrer un concept à des utilisateurs probablement spécialistes dans d'autres domaines. Afin d'insérer des cartes et les couches générées sous QGis, l'environnement de développement en javascript `openlayers` semble idéal puisque fournissant la capacité à charger des couches dans tous les formats qui nous intéressent – WM(T)S, WFS ou GPX pour les traces GPS. Nous avons initialement été sensiblisés à `openlayers` comme environnement de développement pour accéder aux couches fournies

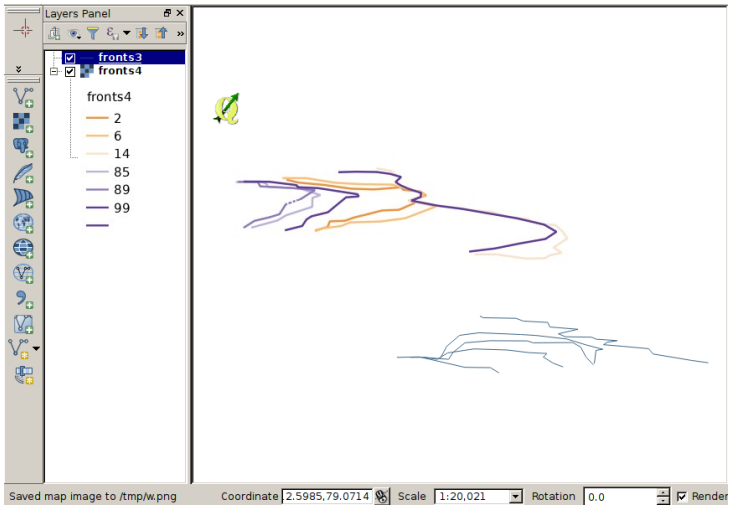

FIGURE 2 – Obtention de couches WM(T)S – images au format bitmap des données (“fronts4”) – et WFS – informations vectorielles (“fronts3”), depuis QGis. Noter le logo de QGis qui est un effet de bord du serveur de test installé par le site web décrivant l'installation de QGis. Effacer `/opt/qgis-server/plugins/HelloServer` ou tout au moins /opt/qgis-server/plugins/HelloServer/icons/icon.png pour s'affranchir de cet effet.

par l'IGN au travers de Géoportail. Nous allons dans un premier temps nous intéresser à la version stable d'Openlayers (version 2) qui semble mature, mais verrons que la nouvelle mouture (version 3) offre des fonctionnalités précieuses qui justifient la migration vers l'environnement en cours de développement.

Une carte Openlayers est formée d'un appel aux bibliothèques javascript (`<script>...</script>`), la création d'une carte remplie d'un fond par défaut – nous choisissons le fond vectoriel OpenStreeMaps (OSM) – et recouvert de diverses couches additionnelles que nous fournirons depuis `qgis-server`.

Dans son expression la plus simple, une page OpenLayers ressemble à

```
<html>
<head>
  <title>Essai de lecture de WMS depuis openlayers</title>
  <script src="http://openlayers.org/api/OpenLayers.js"></script>
</head>
<body>
  <div style="width:100%;height:100%" id="map"></div>
  <script defer="defer" type="text/javascript">
  var map=new OpenLayers.Map('map');
  var wms=new OpenLayers.Layer.WMS("Basic map","http://vmap0.tiles.osgeo.org/wms/vmap0",{layers: 'basic'} );
  map.addLayer(wms);
  map.setCenter(new OpenLayers.LonLat(12,79),9);
</script>
</body>
</html>
```

et tant que nous n'insérons que des couche WMS – donc des images (puisque les données, même vectorielles, sont transmises sous forme d'images avec une résolution adaptée au facteur de grossissement), tout se passe bien. Nous pouvons ainsi ajouter une image Landsat accessible sous forme WMS, voir même un des fronts de glacier :

```
<html>
<head>
  <title>Essai de lecture de WMS depuis openlayers</title>
  <script src="http://openlayers.org/api/OpenLayers.js"></script>
</head>
<body>
  <div style="width:100%;height:100%" id="map"></div>
  <script defer="defer" type="text/javascript">
```

```
    var map=new OpenLayers.Map('map');
    var wms=new OpenLayers.Layer.WMS("Basic map","http://vmap0.tiles.osgeo.org/wms/vmap0",{layers: 'basic'} );
    var dm_wms2013=new OpenLayers.Layer.WMS(
       "Landsat image 2013",
       "http://qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi",
       {layers: "2013_0830utm33n", transparent:"true", format:"image/png" },
       {isBaseLayer: false}
    );

    var fronts1=new OpenLayers.Layer.WMS(
       "Glacier fronts1",
       "http://qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi",
       {layers: "fronts4", transparent:"true", format:"image/png"},
       {isBaseLayer: false}
    );

    map.addLayer(wms);
    map.setCenter(new OpenLayers.LonLat(12,79),9);
    map.addLayer(dm_wms2013);
    map.addLayer(fronts1);
</script>
</body>
</html>
```

Le lecteur est encouragé à compléter cet exemple en ajoutant un front additionnel, par exemple la couche nommée `fronts5`, et observer la conséquence d'échanger l'ordre d'affichage des couches. Ces exemples sont fortement inspirés des excellents tutoriaux fournis par `http://demo.3liz.com/wfst/wfs.html`.

Pour WFS, le problème se corse : le principe de Same Origin impose que le serveur de données soit sur le même site que le serveur web. Ceci est valable pour toute donnée vectorielle. Afin d'illustrer ce problème, nous proposons trois sources de données : un même fichier GPX stocké sur `jmfriedt.sequanux.org`, `jmfriedt.free.fr`, puis `qgis.sequanux.org`. Le serveur `qgis-serveur` fournissant les couches WFS est exécuté comme service Apache2 sur `qgis.sequanux.org`, tandis que la page web qui fait appel à ce service peut être placée sur chacun de ces trois serveurs. Fig. 3 illustre les interdictions d'accès rencontrées, telle qu'indiquées par Firebug : le fichier GPX stocké sur `jmfriedt.sequanux.org` peut être lu par la page `http://jmfriedt.sequanux.org/jmfwfs.html` mais est rejeté par la même page stockée sur `http://jmfriedt.free.fr/jmfwfs.html`. Cette dernière n'a accès qu'au même fichier GPX placé dans le répertoire du serveur `jmfriedt.free.fr`.

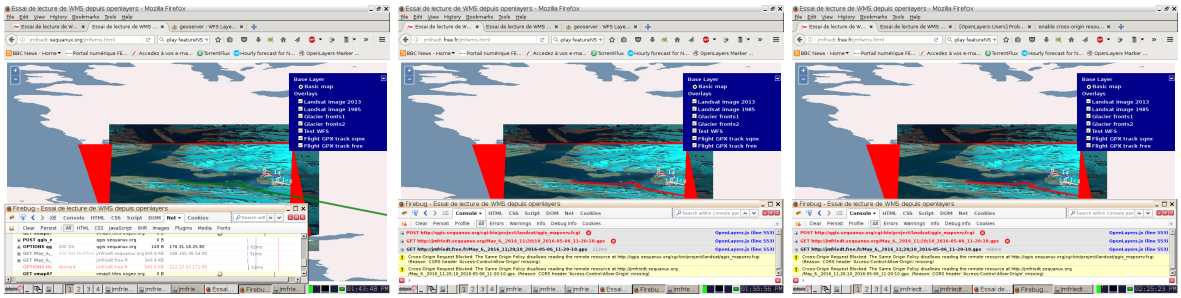

FIGURE 3 – Trois illustrations du problème de Same Origin pour se convaincre de la source du problème.

Ces exemples sont obtenus, après la définition de la carte, par l'appel aux couches vectorielles aux formats GPX ou WFS, de la forme

```
var lgpxfree = new OpenLayers.Layer.Vector("GPX track free", {
  strategies: [new OpenLayers.Strategy.Fixed()],
  protocol: new OpenLayers.Protocol.HTTP({
    url: "http://jmfriedt.free.fr/May_6,_2016_11;20;10_2016-05-06_11-20-10.gpx",
    format: new OpenLayers.Format.GPX()
  }),
  style: {strokeColor: "red", strokeWidth: 5, strokeOpacity: 0.8}
```

```
});

var lgpxsqnx = new OpenLayers.Layer.Vector("GPX track sqnx", {
  strategies: [new OpenLayers.Strategy.Fixed()],
  protocol: new OpenLayers.Protocol.HTTP({
    url: "http://jmfriedt.sequanux.org/May_6,_2016_11;20;10_2016-05-06_11-20-10.gpx",
    format: new OpenLayers.Format.GPX()
  }),
  style: {strokeColor: "green", strokeWidth: 5, strokeOpacity: 0.8}
});

var fronts=new OpenLayers.Layer.Vector("Test WFS", {      // WFS
  strategies: [new OpenLayers.Strategy.BBOX()],
  protocol: new OpenLayers.Protocol.WFS({
    url : "http://qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi",
    featureType : "fronts5",
  }),
});

  map.addLayer(wms); map.setCenter(new OpenLayers.LonLat(12,79),9);
  map.addLayer(fronts);
  map.addLayer(lgpxsqnx);
  map.addLayer(lgpxfree);
```

qui fait appel au même fichier GPX stocké sur deux serveurs (afin de valider que seule la couche sitée sur le même serveur que la page web appelante est acceptable), et la couche nommée “fronts5” fournie au format WFS par `qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi`.

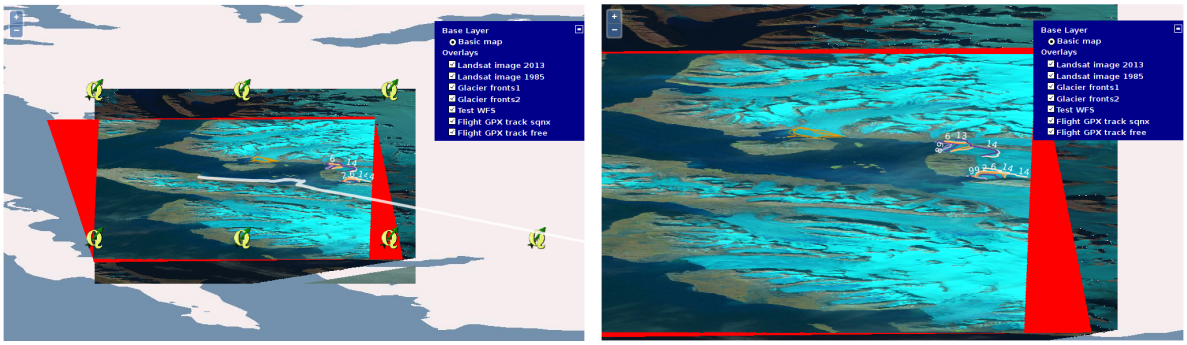

FIGURE 4 – En local (127.0.0.1), tout se passe bien, les données et le serveur sont au même endroit. L’intérêt en terme de diffusion reste limité : placer toutes ses données sur le même site que le serveur résout les problèmes de Same Origin : l’exemple de droite est accessible à `qgis.sequanux.org/jmfwms.html`.

Ayant dépassé tous les eccueils de configuration, nous avons finalement la satisfaction de voir toutes les couches s’afficher sur la même carte, en ajoutant l’onglet de commandes qui permet d’activer ou désactiver chaque couche afin d’aisément comparer la position des fronts des glaciers et leur évolution dans le temps :

```
var map=new OpenLayers.Map('map');
ls=new OpenLayers.Control.LayerSwitcher({'div':OpenLayers.Util.getElement('layerswitcher')});
map.addControl(ls);
ls.maximizeControl();
```

Il n’aura cependant pas échappé au lecteur que nos images satellites semblent fortement déformées (comparer Fig. 1 et Fig. 4). Il s’agit ici de la conséquence de la projection de Mercator, qui tend à considérablement grossir l’axe des abscisses lorsque la zone considérée s’éloigne de l’équateur. Quoique acceptable jusqu’à des latitudes de $\pm 60°$, l’effet devient vraiment significatif au-delà et carrément désagréable par 79°N. C’est pourquoi les projections locales, tentant de trouver un plan localement tangent à la sphère représentant le globe terrestre, ne prétendent pas à s’appliquer à l’ensemble de la Terre mais uniquement à un petit segment de longitude. Malheureusement, Openlayers 2 ne supporte pas la reprojection à la volée des données bitmap, nous sommes coincés avec la projection imposée depuis Google Maps qui continue à déformer les pôles. Nous allons remédier à ce problème en passant sous Openlayers 3 et retrouver ainsi les belles cartes de QGis.

4 À la pointe de la technologie : `openlayers 3`

Dans la course aux fonctionnalités, nous ne cessons de casser ce qui marche pour mettre à jour les nouvelles fonctionnalités. Pourquoi donc casser ce bel exemple Openlayers2 pour le remplacer par Openlayers3 ? La déformation excessive des cartes aux latitudes élevées est un défaut bien connu de la projection de Mercator utilisée par défaut par tous les environnements cartographiques de la toile (projection de code 900913 – l'écriture leet de Google – puisque la norme choisie n'a pas été validée par l'instance qu'est l'EPSG). Étant conscients des déficiences de la projection de Mercator, nous voudrions pouvoir sélectionner un mode de projection localement tangent à la région considérée – à savoir Universal Transverse Mercator (UTM). Et nous en arrivons à l'excuse pour passer de Openlayers 2 à 3 : la version actuelle d'Openlayers ne permet pas la projection au vol des images, et impose donc une sortie déformée, particulièrement gênante lorsque nous nous éloignons de l'équateur.

Nous reprenons donc toutes nos investigations, mais cette fois dans le contexte d'Openlayers 3, avec la nécessité pour chaque nouvelle couche de préciser le mode de projection d'entrée (toutes nos données sont stockées en cordonnées sphériques, donc WGS84 de code EPSG :4326. Openlayers ne connaît pas tous les modes de projection de sortie : nous devons les définir selon nos besoins (Fig. 5).

FIGURE 5 – Gauche : le passage à une projection locale WGS84/UTM33N résout la déformation observée auparavant dans Openlayers 2 et son Google Mercator. Cependant, cette projection n'a pas prétention de s'étendre à tout le globe, et réduire le grossissement met en évidence la déformation sur les régions adjacentes, ici le nord Canada et la Sibérie (droite). Cet exemple est disponible à `jmfriedt.sequanux.org/reproj.html`

Heureusement, `proj4` se charge de cette opération pour nous, et `http://spatialreference.org/ref/epsg/wgs-84-utm-zone-33n/` nous informe de la façon de définir WGS84/UTM33N (la bande qui couvre la Norvège – nous aurions choisi UTM31N pour la France). Ainsi, le mode de projection qui va nous intéresser se définit par

```
proj4.defs('EPSG:32633','+proj=utm +zone=33 +ellps=WGS84 +datum=WGS84 +units=m +towgs84=0,0,0 +no_defs');
```

et nous pouvons faire appel aux outils de conversion de la forme

```
var osmLayer = new ol.layer.Tile({ source: new ol.source.OSM() });

var map = new ol.Map({
  controls: ol.control.defaults().extend([new ol.control.ScaleLine()]),
  target: 'map',
  view: new ol.View({projection: 'EPSG:32633',center: ol.proj.fromLonLat([10, 79], 'EPSG:32633'),zoom: 9})
});

var dm_wms2013 = new ol.layer.Tile({
  preload: Infinity,
  visible: true,
  source: new ol.source.TileWMS(({
    url: 'http://qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi',
    params: {'LAYERS': '2013_0830utm33n', 'TILED': true, 'VERSION': '1.3.0',
    'FORMAT': 'image/png8', 'WIDTH': 256, 'HEIGHT': 256, 'CRS': 'EPSG:4326'},
    serverType: 'geoserver'
  }))
});

...
```

```
map.addLayer(osmLayer);
map.addLayer(dm_wms2013);
```

dans lequel la projection de la carte respecte la norme EPSG (WGS84/UTM33N s'appelle EPSG :32633) que nous avons défini par `proj4` auparavant, tandis que les données en entrée (ici issue de la fonction WM(T)S du serveur) sont au format WGS84 en coordonnées sphériques (aussi nommée EPSG :4326).

5 Ajout automatique de toutes les couches WMS d'un projet `qgis-server` dans `openlayers2`

Une dernière question que nous pouvons nous poser, dans le contexte d'un travail collaboratif où chaque utilisateur est susceptible d'ajouter ses propores couches, tient en la capacité à automatiser la génération d'une page web contenant toutes les couches fournies par un serveur QGis. Cela signifie donc récupérer la description XML des capacités du serveur, découper cette description pour extraire le nom des couches disponibles, et finalement générer automatique le script `javascript` affichant toutes ces couches. De telles fonctionnalités sont fournies (paquet `php-xml` dans la distribution Debian) par `simplexml`. Ainsi, un premier script qui se contente de lister toutes les couches disponibles en recherchant d'abord les fonctionnalités WMS du serveur, puis en étudiant [3] la liste des propriétés (`FeatureTypeList`) et finalement les propriétés de chacun de ces objets (`FeatureType`) pour en extraire le nom (`FeatureTypeList->FeatureType->Name`) ressemble à

```
<html><head><title>XML depuis PHP</title></head><body>
<?php
 if (extension_loaded('simplexml')) {
  echo "extension installed:<hr>";
  $mypix = simplexml_load_file(urlencode(
   'http://127.0.0.1/cgi-bin/project/landsat/qgis_mapserv.fcgi?SERVICE=WFS&VERSION=1.0.0&REQUEST=GetCapabilities'));
  foreach ($mypix->FeatureTypeList as $pixinfo):
   foreach ($pixinfo->FeatureType as $nam):
    if ($nam->Name!='') echo $nam->Name,"<br>";
   endforeach;
  endforeach;
 }
 else echo "missing extension<br>";
?>
</body></html>
```

Ayant compris ce mécanisme, il devient simple de générer les champs décrivant chaque couche WMS affichée, en lui attribuant le même nom de couche que celui utilisé dans QGis. Le résultat est

```
<html><head><title>Toutes les couches WMS depuis openlayers</title>
  <script src="http://openlayers.org/api/OpenLayers.js"></script>
</head>
<body>
 <div style="width:100%;height:100%" id="map"></div>
 <script defer="defer" type="text/javascript">
 var map=new OpenLayers.Map('map');
 ls=new OpenLayers.Control.LayerSwitcher({'div':OpenLayers.Util.getElement('layerswitcher')});
 map.addControl(ls);
 ls.maximizeControl();
 var wms=new OpenLayers.Layer.WMS("Basic map",
     "http://vmap0.tiles.osgeo.org/wms/vmap0",{layers: 'basic'} );
<?php
 $k=1;
 if (extension_loaded('simplexml')) {
  $mypix = simplexml_load_file(urlencode(
   'http://127.0.0.1/cgi-bin/project/landsat/qgis_mapserv.fcgi?SERVICE=WFS&VERSION=1.0.0&REQUEST=GetCapabilities'));
  foreach ($mypix->FeatureTypeList as $pixinfo):
   foreach ($pixinfo->FeatureType as $nam):
    if ($nam->Name!='')
    {echo "var dm_",$k,"=new OpenLayers.Layer.WMS(\"",$nam->Name,"\",";
     echo   "\"http://127.0.0.1/cgi-bin/project/landsat/qgis_mapserv.fcgi\",";
     echo   "{layers: \"",$nam->Name,"\",";
```

3. les divers champs se déduisent de l'analyse de la sortie de `wget-O-"http://127.0.0.1/cgi-bin/mon_projet/qgis_mapserv.fcgi?SERVICE=WFS&VERSION=1.0.0&REQUEST=GetCapabilities"`

```
     echo   "transparent:\"true\", format:\"image/png\"},{isBaseLayer: false});\n";
     $k++;
    }
   endforeach;
  endforeach;
  echo "map.addLayer(wms);\n";
  echo "map.setCenter(new OpenLayers.LonLat(12,79),9);\n";
  for ($x=1;$x<$k;$x++) {echo "map.addLayer(dm_$x);\n";}
 } else echo "missing extension<br>";
?>
</script></body></html>
```

qui donne la Fig. 6. Il serait d'une part très fastidieux d'ajouter à la main toutes les couches de ce projet, mais surtout dans cet exemple l'affichage s'adapte automatiquement aux nouvelles couches ajoutées par les divers contributeurs au projet.

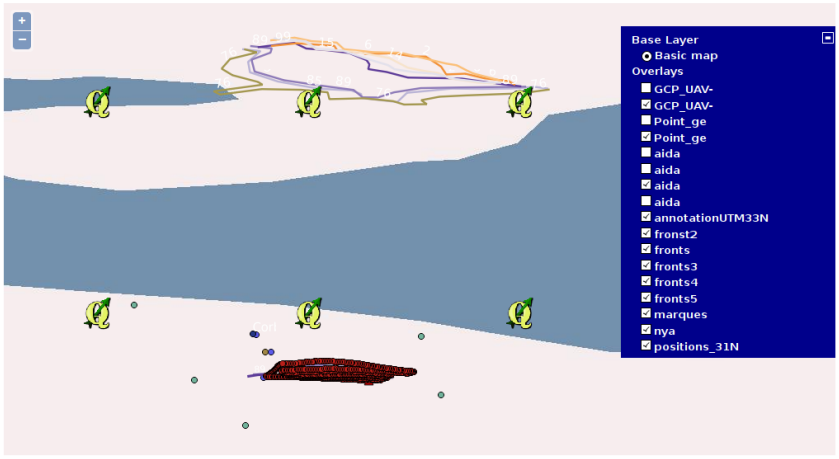

FIGURE 6 – Ajout automatique de toutes les couches fournies par un serveur WMS sur dans un contexte d'Openlayers2. Le script d'exemple `/opt/qgis-server/plugins/HelloServer` qui affiche le logo que QGis a été volotairement laissé en place pour distinguer le serveur en production (`qgis.sequanux.org`) du serveur de tests (127.0.0.1).

6 Ajout d'images orthorectifiées produites par microdrone

Nous avions présenté le flux de traitement permettant de générer une série d'images orthorectifiées et de modèles numériques d'élévation associés [2] avec le soucis de comparer ces jeux de données entre eux, mais sans prétention de les positionner dans un référentiel absolu partagé par convention entre plusieurs utilisateurs. Il s'avère [5] que la solution de simplement translater un jeu de données par rapport à l'autre ne suffit plus lorsque la zone considérée s'étend sur plusieurs hectares. Nous avons observé que dans ce cas, le positionnement par GPS (mono-fréquence, tel que fourni sur microdrone DJI Phantom3 Professional) n'est exact qu'à une dizaine de mètres, alors que nous visons un positionnement au décimètre près pour des comparaisons de vols successifs. Il s'avère que vouloir corriger les défauts d'échelle et de position du modèle numérique d'élévation en lui appliquant les corrections permettant de superposer les images orthorectifiées est une approche peu judicieuse, et qu'il vaut mieux insérer dans le flot de traitement l'étape d'exploitation des points de contrôle au sol pour corriger les défauts du modèle de caméra et de leur position [4]. Pour notre part, les points de contrôle au sol (GCP) sont acquis à posteriori dans Google Earth – les coordonnées sphériques fournies avec 6 décimales sont précises à 11 cm à l'équateur : 7 points aisément reconnaissables dans les jeux de données sont d'une part identifiés dans Google Earth (bandes de parking, barrière, coin à la base de batiments) avec des coordonnées converties de coordonnées sphériques (WGS84) vers un référentiel projeté localement tangent à la Terre (WGS84/UTM31N), et d'autre part leur position (en pixel) identifiée sur les images acquises par drone. Ce couple de fichier

4. `http://forum-micmac.forumprod.com/campari-residuals-differ-from-the-point-cloud-t881.html`

(coordonnées dans l'espace v.s coordonnées sur les photographies) alimente l'outil `campari` de Micmac. Après traitement dans ces conditions, nous observons une exactitude de position par rapport aux couches vectorielles de Openstreetmaps mais surtout par rapport à une image aérienne de l'IGN (BD ORTHO®, convertie de Lambert93 à WGS84/UTM31N puisque nous avons vu que Openlayers 2 ne sait pas le faire à la volée) meilleure qu'une vingtaine de centimètres – erreur attribuable à notre manque d'assiduité à pointer les points de contrôle. Ce jeu de données est disponible à `http://qgis.sequanux.org/cgi-bin/project/femto/qgis_mapserv.fcgi`.

FIGURE 7 – Résultat de l'insertion de trois images orthorectifiées, en comparaison de données de référence que sont les images aériennes de la BD ORTHO de l'IGN et les données vectorielles de Opensteetmaps. En haut à gauche : deux images orthorectifiées acquises à 30 minutes d'intervalle positionnées au moyen de 7 GCPs. En haut à droite : le même jeu de données, positionné uniquement par les positions GPS du drone au moment de la prise de vue. Le trait rouge, indiquant la différence de position, est long de 9,8 m. En bas à gauche : superposition d'une image produite par drone (moitié gauche) avec une image IGN (moitié droite). En bas à droite : comparaison de deux orthophotos produites à un mois d'intervalle.

7 Conclusion

Nous avons proposé un environnement de travail permettant de disséminer des informations géoréférencées soit vers des utilisateurs de logiciels dédiés de gestion de systèmes d'informations géographiques (SIG) ou vers les API web associées (openlayers). Est-ce que ce mode de dissémination est réellement utilisé en pratique ? L'institut polaire norvégien (NPI) propose ses informations dans ce format tel que décrit à `http://geodata.npolar.no/#basemap-services`. Ainsi, sous QGis, configurer l'onglet WM(T)S vers `http://geodata.npolar.no/arcgis/rest/services/Basisdata/NP_Ortofoto_Svalbard_WMTS_25833/MapServer/WMTS?` donne accès aux informations bitmap que sont les photographies aériennes de très haute résolution (jusqu'à 16 cm/pixel !) tandis que l'url `http://geodata.npolar.no/arcgis/services/CryoClim/glaciers/MapServer/WmsServer?` dans l'onglet WFS donne accès aux informations vectorielles que sont les limites des glaciers extraites des images satellitaires SPOT.

Le lecteur est encouragé à reproduire ces expériences sur ses propres zones d'intérêt. Compte tenu de la résolution médiocre de Landsat, le résultat est peu convaincant pour la vallée de Chamonix, mais les glacier groenlandais (Illulisat près de la baie de Disco) ou la banquise antarctique sont parfaitement

appropriés à cette démonstration, voir la mer d'Aral [5] (un peu pénible à assembler car de superficie supérieure à ce que le site de l'USGS permet d'analyser) ou la mer morte [6] pour laquelle l'extension des bassins d'exploitation du sel est particulièrement visible. Le lecteur saura-t-il calculer la surface ainsi affectée ? Indice : tracer un polygone avec l'outil *New Shapefile* puis dans la table des attributs, créer un nouveau champ pour chaque polygone avec la variable `$area`.

Une autre voie d'étude qui semble intéressante dans le cadre de la diffusion sur le web de données géoréférencées est `qgis2web`, greffon de QGis actuellement inexploitable sous Debian testing/sid compte tenue d'une dépendance cassée.

Finalement, étant donné que toutes les informations pertinentes à une utilisation en surface sont disponibles sur OpenStreetMaps, ne serait-il pas temps de commencer à considérer la cartographie des services sous-terrains (eau, gaz, électricitié), qui même s'ils sont propriété de leurs exploitants respectifs, sont devenus des services nécessaires au quotidien d'un habitant d'Europe occidentale actuel : chaque kilomètre de route contient plusieurs dizaines de kilomètres de services sous terrains [6] qui ne demandent qu'à être cartographiés à l'occasion des ouvertures de routes.

Remerciements

J.-P Culas (CM-Drones, Besançon) a effectué les vols au-dessus du parking de FEMTO-ST. Les membres du forum Micmac (`http://forum-micmac.forumprod.com/`) ont une fois de plus partagé leurs connaissances pour corriger les défaillances dans les traitements que nous proposions initialement.

Références

[1] M. Pierrot-Deseilligny, J.-M Friedt, *La photogrammétrie pour tous : MicMac pour la reconstruction 3D de scènes géoréférencées à partir de photographies numériques*, turorial à FOSS4G-fr 2016, décrit à `jmfriedt.free.fr/foss4g_2016`

[2] J.-M. Friedt, F. Tolle, É. Bernard, *Utilisation de Micmac pour la génération de modèle numérique d'élévation par traitement d'images acquises par microdrone*, GNU/Linux Magazine France **191**, pp.48–57 (Mars 2016)

[3] S. Erle, R. Gibson & J. Walsch, *Building the geospatial web*, dans *Mapping Hacks – Tips & Tools for Electronic Cartography*, O'Reilly (2005)

[4] R. Gibson & S. Erle, *Google Maps Hacks – Tips & Tools for Geographic Searching and Remixing*, O'Reilly (2006)

[5] J. Lisein, N. Pineux, M. Pierrot-Deseilligny, A. Degré & P. Lejeune, *Détection de l'érosion dans un bassin versant agricole par comparaison d'images multidates acquises par drone*, Colloque *Drones et moyens légers aéroportés d'observation*, 26/06/2014 (Montpellier), disponible à `http://orbi.ulg.ac.be/handle/2268/171616`

[6] chaque kilomètre de route à Hong-Kong recouvre 47 km de services enterrés, `http://www.uti.hk/media/attachments/2nd_ICUMAS_full_paper.pdf` ou `http://www.scmp.com/news/hong-kong/article/1647088/small-army-utility-specialists-keeps-hongkongers-safe-pipes-cables`, ou en d'autres termes il existe 47 cables, tuyaux, fibres optiques et autres modes de transmission de fluides et d'informations sous chaque route.

5. `http://www.nasa.gov/mission_pages/landsat/news/40th-top10-aralsea.html`
6. `http://earthobservatory.nasa.gov/IOTD/view.php?id=77592`