Laboratory of Geographic Information Systems

Working with KML data in OpenLayers

1. Introduction to OpenLayers 6 and coordinate capture

This exercise uses version 6.5.0 of the OpenLayers library, which is a free tool for creating Web based Geographic Information Systems.

To begin with, open the file ***OpenLayersKML.html*** in your favourite text editor (eg. Notepad). The first noticeable element of this file is the list of imported styles and scripts. It is followed by the <*body*> segment, which contains several <*div*> elements which will be used during this exercise. It is followed by the <*script*> segment, which contains initialization code for the OpenLayers library *map* element. Initially, the script creates a page which displays a map obtained from the OpenStreetMap service (*layers: [osm]*). Other parameters of the *ol.Map({...})* constructor include:

- *target: html-element* - the object on the page where the map will be displayed (the elements are accessed through the javascript object method *document.getElementById ("divID"))*,

- *view: value*, where value is an object of type *ol.View({...})* containing parameters such as:

- *center:* the point at which the map is initially centered, and

- *zoom:* the initial map zoom level.

In order to add the coordinate capture functionality to the map, it is necessary to create appropriate places for holding them. To do this, create two *input* objects with id's of „lonbox" and „latbox" in the body of the HTML document. Then capture the event of clicking on the map using the method *map.on("singleclick", function(event) {<insert_code_here>}).* This method registers the *function(event)* as the object which reacts to the user clicking the map. Next, inside the *function* body, read the captured coordinates from the *event* by using *map.getEventCoordinate(event.originalEvent)* and save them to a new variable. The returned value contains a two-dimensional table holding two coordinates in the Mercator system. Then, transform the coordinates inside the variable to WGS84 using the method *ol.proj.toLonLat(variable_name)*. Finally, assign the values of elements [0] and [1] of your variable to the objects *lonbox* and *latbox* respectively (you can use the *document.getElementById('...').value* method).

2. Basics of geographic object description using Keyhole Markup Language.

Keyhole Markup Language is a language for description of spatial objects based on XML. Its most important elements are:

1. Document – a container for objects and styles. Required if using shared styles.
2. Placemark – object representing a given geographic point feature.
3. LineString – three-dimensional object representing a polyline. Consists of a series of points defined by their longitude, latitude and altitude (above sea level).
4. Polygon – three-dimensional object defined by two closed LinearRing shapes, which define its OuterBoundary and InnerBoundary.

Alongside the provided html file you'll find sample KML files containing the aforementioned data types.

Adding KML files to the webpage

KML files may be added to a webpage created with OpenLayers in the form of vector layers. A sample KML vector layer is created in the following way:

```
var layer_name = new ol.layer.Vector({
    source: new ol.source.Vector({
        url: "kml/filename.kml",
        format: new ol.format.KML()
    })
})
```

After being created, a vector layer must be added to the map by using:

```
map.addLayer( layer_name );
```

Turning off CORS for local data

Modern web browsers use a mechanism called Cross-Origin Resource Sharing (CORS) for controlling whether certain elements of the web page may be requested from another domain. While this makes a lot of sense for online data, the latest versions of the mechanism are overly sensitive, treating local files from the same directory as coming from a different domain. Usually the only workarounds are to either disable CORS entirely, or put the files on a web server. Luckily, the Firefox web browser provides an option for disabling CORS for local files only, which allows for completing this exercise without using a web server, and does not impact browser security in a meaningful way. To do this, input *about:config* in the Firefox address bar, then search for *privacy.file_unique_origin* (or *security.fileuri.strict_origin_policy* in more recent versions of the browser) and change its value to *false*.

Displaying KML information on pointer move

KML files may contain attributes and descriptions, which are not displayed on the map by default. In order to display this data on the webpage, we will use two methods. The first one, *pointermove*, will be called every time the user moves the mouse. Inside this method we will call the second one, *displayFeatureInfo*, which will display a description of KML layers. The methods look like this:

```
map.on("pointermove", function(evt) {
    if (evt.dragging) {
        return;
    }
    var pixel = map.getEventPixel(evt.originalEvent);
    displayFeatureInfo(pixel);
});

var displayFeatureInfo = function(pixel) {
    var features = [];
    map.forEachFeatureAtPixel(pixel, function(feature) {
```

```
            features.push(feature);
    });
    if (features.length > 0) {
        var info = [];
        var i, ii;
        for (i = 0, ii = features.length; i < ii; ++i) {
            info.push(features[i].get("name"));
        }
        document.getElementById("infor").innerHTML = info.join(", ") ||
"(unknown)";
        map.getTarget().style.cursor = "pointer";
    } else {
        document.getElementById("infor").innerHTML = " ";
        map.getTarget().style.cursor = "";
    }
};
```

Attention! The above code contains an error (the same one) in two different places! The error needs to be corrected in order for the code to work as intended.

When the code works correctly, a description of the KML object being pointed at by the mouse cursor will be displayed in the lower left corner of the web page (below the map area).

3. Reverse Geocoding

The process of geocoding involves assigning geographic coordinates to a given location. The process of obtaining a location name for a set of coordinates is called reverse geocoding. There are many reverse geocoding services available. Depending on the business model, licenses, transfer restrictions, etc., full access to their functionalities may require payment. During the exercise, we will use the Nominatim database from OpenStreetMap, which uses a publicly exposed API. An example call of Nominatim's reverse geocoding API is:

https://nominatim.openstreetmap.org/reverse?format=json&lon=18&lat=54

The above query returns a description of the location denoted by the lon / lat coordinate pair. A sample code adding a method for reverse geocoding of a given point may be found below.

```
function simpleReverseGeocoding(lon, lat) {
    var addr = "https://geocode.maps.co/reverse?lon="+lon+"&lat="+lat;
    fetch(addr)
     .then(function(response) {
            return response.json();
    })
    .then(function(json) {
     // display result
    });
}
```

In order to display the geocoding results, a popup element will be created on the map. First, create an appropriate * * element in the HTML body, right below * *:

```
<div id="popup" class="ol-popup"></div>
```

Next, insert the following lines below the map initialization code (*var map = new ol.Map(...)*):

```
var popupBody = document.getElementById('popup');
var popup = new ol.Overlay({
     element: popupBody,
});
map.addOverlay(popup);
```

Afterwards, in the *display result* section of *simpleReverseGeocoding* add the following lines:

```
  var coordinates = ol.proj.fromLonLat([lon, lat]);
  popup.setPosition(coordinates);
  document.getElementById("popup").innerHTML = json.display_name;
```

Finally, in the <style> section add the following:

```
.ol-popup {
    position: absolute;
    background-color: white;
    box-shadow: 0 1px 4px rgba(0,0,0,0.2);
    padding: 15px;
    border-radius: 10px;
    border: 1px solid #cccccc;
    bottom: 12px;
    left: -50px;
    min-width: 280px;
 }
.ol-popup:after, .ol-popup:before {
    top: 100%;
    border: solid transparent;
    content: " ";
    height: 0;
    width: 0;
    position: absolute;
    pointer-events: none;
  }
```

```
.ol-popup:after {
    border-top-color: white;
    border-width: 10px;
    left: 48px;
    margin-left: -10px;
 }
.ol-popup:before {
    border-top-color: #cccccc;
    border-width: 11px;
    left: 48px;
    margin-left: -11px;
 }
```

The only remaining task is to call the simpleReverseGeocoding function when the user clicks on a given location. For this purpose, you can use the existing *map.on("singleclick",...)* function.

If the above tasks have been completed correctly, clicking on any element of the map will now produce a popup containing the geographic address of this place.

TASKS TO BE CARRIED OUT

1. According to the given instructions, build a web page which allows for capture of geographic coordinates and display of KML files via the OpenLayers API. 1.0 pt.
2. Display on the map a KML file representing at least 5 objects near your place of residence (eg. Bus stops, clubs, cafes). The objects should be assigned short descriptions. The points should be marked using KML ***"Placemark"*** tags. 1.0 pt.
3. Prepare and display a polyline (with at least 5 segments)representing the road from the place of your residence to the University. People living in dormitories should mark the approximate route of the road / railway from their home town. As a reference, you can use the attached Polyline_example.kml 1.0 pt.
4. Draw a chosen building of the Gdansk University of Technology (with at least six vertices!) using the KML ***Polygon*** tag. As a reference, you can use the attached Polygon_example.kml 1.0 pt.
5. Present the Reverse Geocoding mechanism in the area of Poland. 1.0 pt.