


Matlab plotting into Google Earth

Contents

- [Contents](#)
- [Getting up-and-running](#)
- [Keyword-value pairs: setproperty, oetnewfun](#)
- [Automating saving to file: fileName](#)
- [Adapting menu appearance: description,snippet,kmlName](#)
- [Adapting vertical scale/exaggeration: zScaleFun](#)
- [Adapting color: cLim,colorMap,colorSteps,CBcolorTitle](#)
- [Adding time: TimeIn, TimeOut](#)

Getting up-and-running

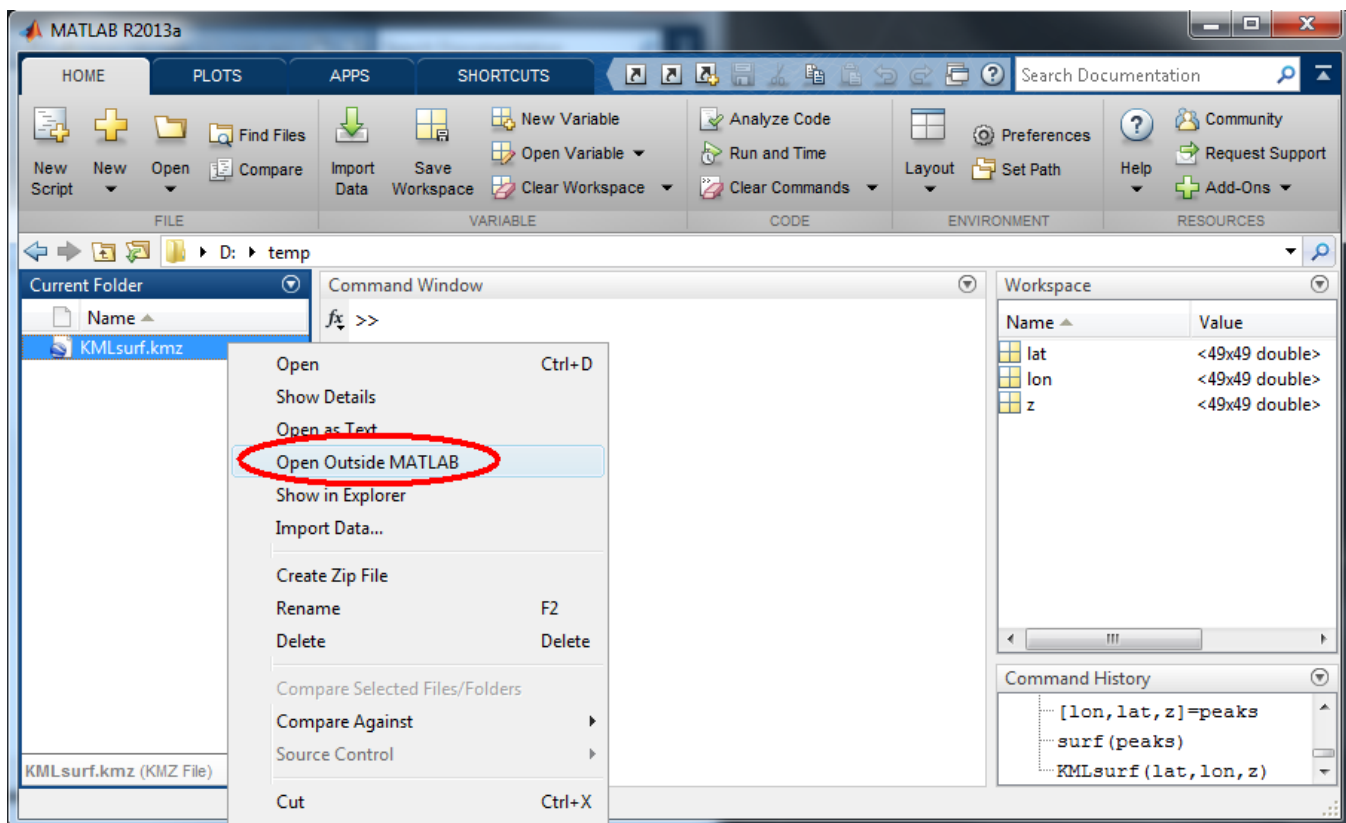
For plotting Google Earth in Matlab we made the GooglePlot toolbox as part of the Matlab OpenEarthTools. To get a copy of this free and open source toolbox, (i) [register for the svn repository](#), (ii) [install the TorToiseSVN svn client for windows, and use that to get a copy of the complete toolbox](#) (only getting the GooglePlot folder won't work due to dependencies), and (iii) activate openearthtools in your Matlab session by adding it to your path with [oetsettin](#)gs. Running this will display a message screen with a clickable link to GooglePlot. Otherwise type `help googleplot`. This will display a list of all available plot types for Google Earth. We attempted to create a version for each regular Matlab plot command by prepending "kml", e.g. `kmlsurf` will plot a surface just like Matlabs own `surf`. We actually created more plot types than matlab itself offers, i.e. `KMLquiver` for curved arrows and `KMLcylinder` for a 3D cylinder.

In this tutorial we will explain the way to use GooglePlot with `KMLmesh` and `KMLsurf`. After `help googleplot` you can mouse-click on `KMLsurf`, or type `help KMLsurf`. The calling of `KMLsurf` is as easy as `surf` itself, simply pass `x,y,z`, and optionally a `c` for the color when this is different than `z`. Note that GooglePlot always plots  LAT before LON, while Matlab uses `x` before `y`. We will use an example dataset that is shipped with Matlab `peaks`.

The following code snippet will generate a `KMLsurf` object.

```
[lon,lat,z]=peaks
surf(peaks)
KMLsurf(lat,lon,z)
```

Note that `KMLsurf` will throw a pop-up menu asking for the filename where to store the kml object, which always has the extension `kmz` or `kml`. Saving data to a file ([KMLsurf.kmz](#) in this case) is the way viewing data in Google Earth work. You can view this file (which has extension `*.kml` or `*.kmz`) in Google Earth by double-clicking it in your regular file explorer. You can also open it from within Matlab by right-clicking the file and choosing "open outside Matlab".



You will see the peaks dataset in The Gulf of Guinea near Africa, where (lat,lon) are in the value range of the peaks dataset. You have now seen the functionality of KMLsurf. The remainder of this wiki is dedicated to make your plots nicer by tailoring it's appearance, and making the process faster. For this you will need to pass extra options to the GooglePlot functions.

Keyword-value pairs: *setproperty*, *oetnewfun*

Each GooglePlot function, and in fact most openearthtools Matlab functions, adhere to a specific convention that allows users to pass extra parameters using keyword-value pairs. This approach is similar to the way Matlab allows users to modify properties of plots, figures and axis. For instance, to change the color and width of a line plot, Matlab specifies this syntax for a thick grey line (RGB value of .5).

```
plot(x,y,'color',[.5 .5 .5],'linewidth',2)
```

You can also ask for a plot handle when you call it, and then ask for all keywords that you can modify with `get`, get all default values with `set`, and also modify a selection with `set`.

```
h = plot(x,y)
get(h)
set(h)
set(h,'color',[.5 .5 .5],'linewidth',2)
```

OpenEarthTools adopted this approach, facilitated by the important core function `setproperty` (we test this function very thoroughly due its central function, in fact other scripting languages provide this function in their core, e.g `kwargs` in python). For tool users, remember our other convention that you can call a GooglePlot *without arguments* to get a list of all available properties. This will generate a list with the default values. You can also *edit* the function to see the struct with all properties. For developers to use this keyword-value functionality, call `oetnewfun` to start a new function using a template.

```

KMLsurf() % call without arguments

ans =
      kmlName: ''
      snippet: ''
    description: ''
      fileName: ''
    ...
      lineWidth: 1
      lineColor: [0 0 0]
    ...
      zScaleFun: @(z)(z+20).*5
    ...
      colorbar: 1
      colorMap: @(m)jet(m)
    colorSteps: 16
      cLim: []
    ...
      timeIn: []
      timeOut: []

```

There are a lot of properties that can be set, so much that you can be overwhelmed, but we reassure you we chose good default values. We will describe some important properties know, the ones highlighted in the code block above.

Automating saving to file: *fileName*

If you use GooglePlot often, the pop-up for a filename is annoying and blocks automation. Keyword 'fileName' allows you to pass the filename to prevent a pop-up. You noticed that KMLsurf was a bit slow. For testing this extra option, it would be nice to have a faster option available. `KMLmesh` also produces a 4D grid, but without color. The extension you specify determines whether you get an kml or a kmz file. The kml file format is a specific xml file, following the open source [OGC KML](#) standard. This is an ascii file, which can become rather big. You can simply zip the kml file, and then rename it to *.kmz. A kmz file is simply an ordinary zip file with a kml inside (plus images that are associated with the plot, in this case the colorbar image). We recommend always to select kmz as extension, to prevent separate colorbar images from appearing in your folder.

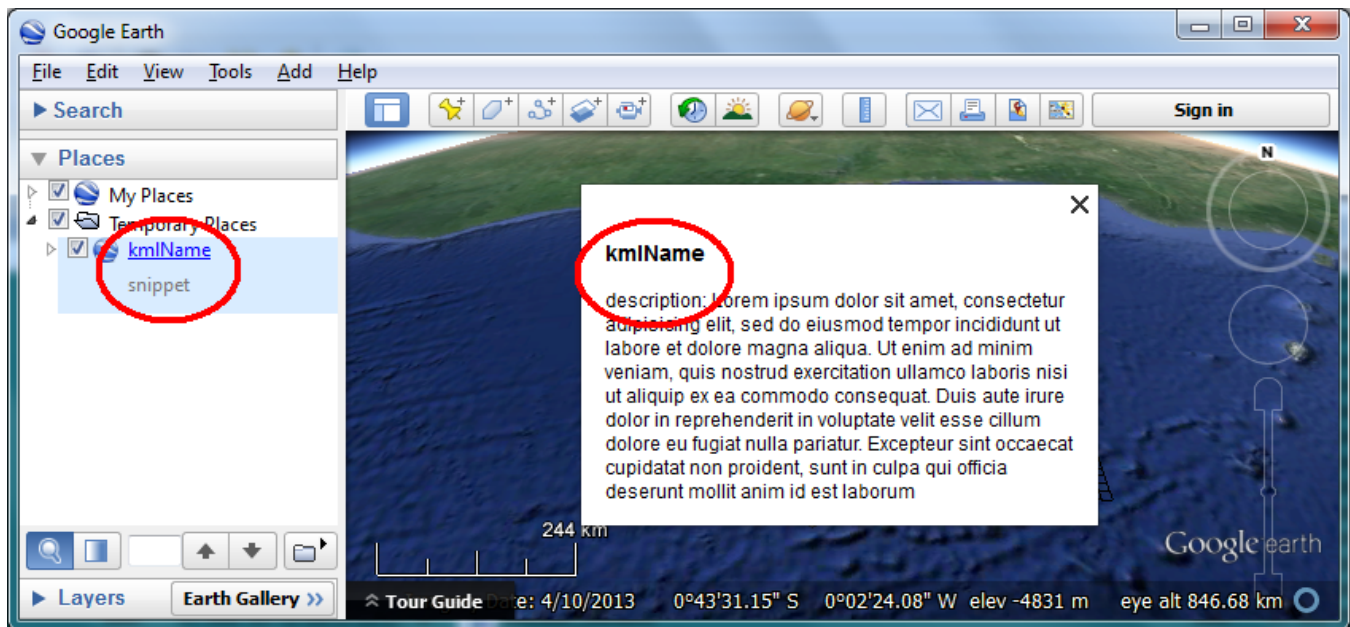
```
KMLmesh(lat,lon,z,'fileName','KMLmesh.kmz')
```

You will notice that the black lines in [KMLmesh.kmz](#) are ill-visible on the blue ocean. Use keyword *lineColor* to set the lines to white, using the rgb triplet [1 1 1](#) to produce [KMLmesh_white.kmz](#).

```
KMLmesh(lat,lon,z,'fileName','KMLmesh_white.kmz','lineColor',[1 1 1])
```

Adapting menu appearance: *description,snippet,kmlName*

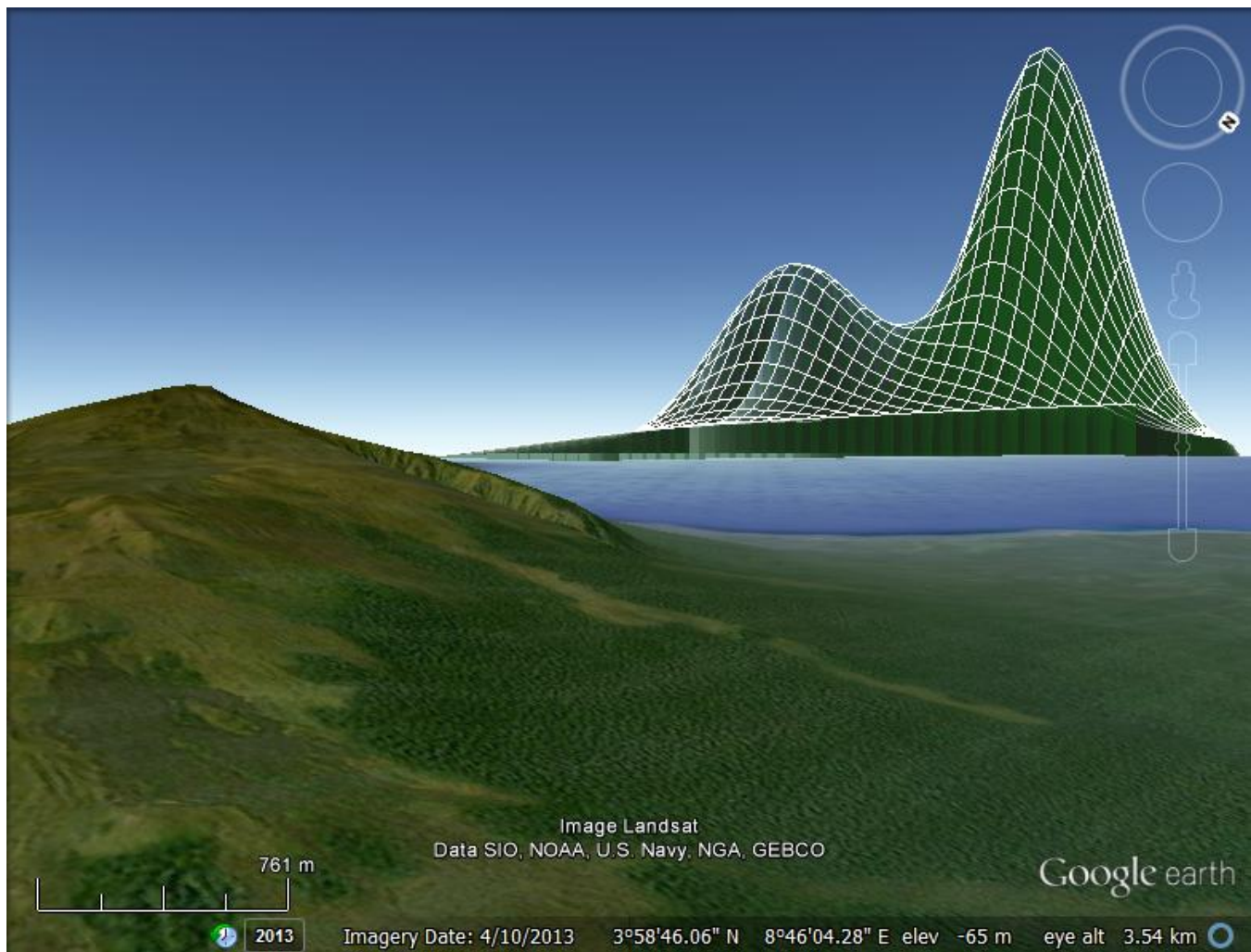
With description, snippet and kmlName you can add text to the left menu, and to a pop-up. You can add html to description to make it look professional.



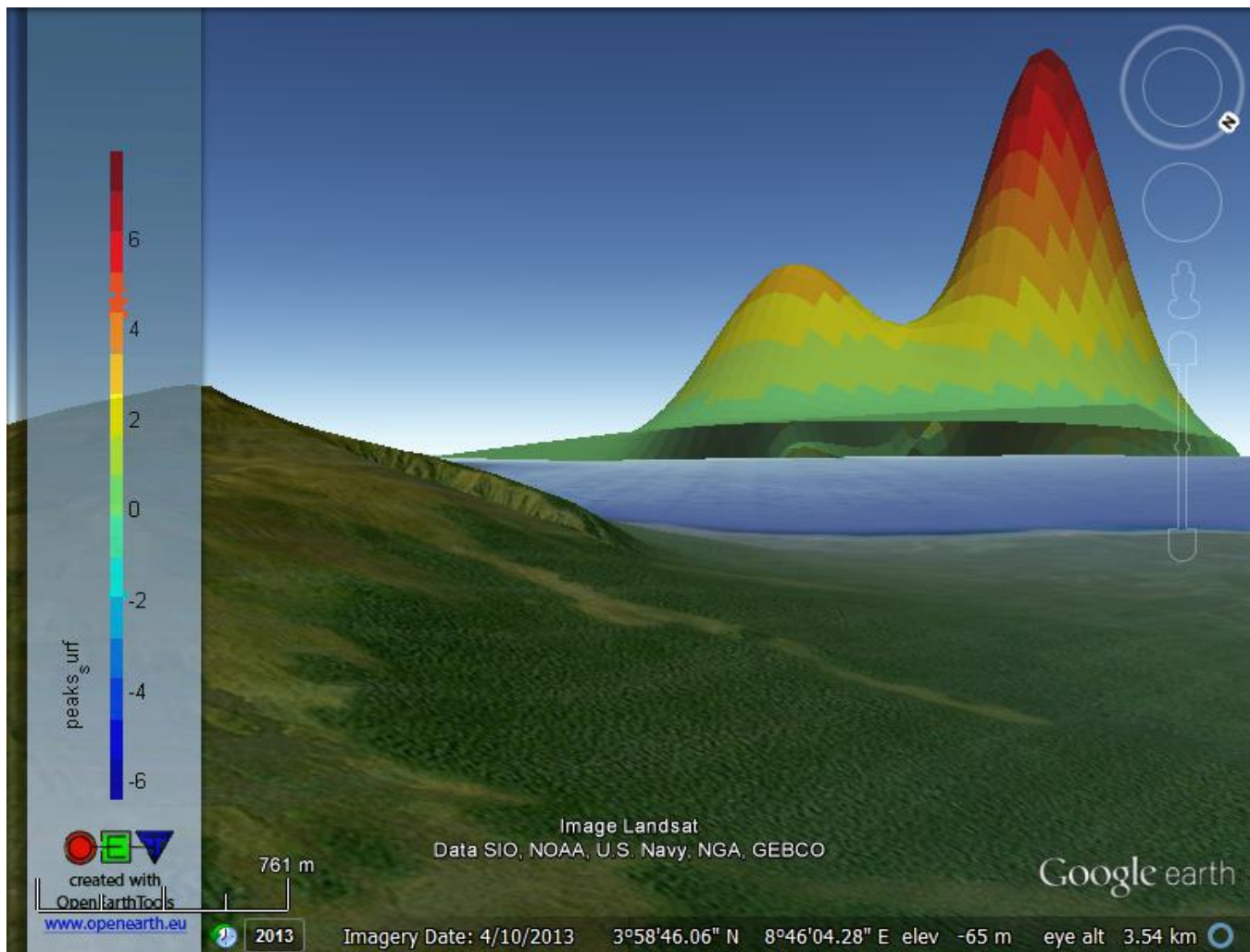
Adapting vertical scale/exaggeration: *zScaleFun*

The KMLsurf and KMLmesh plots look like regular pcolor plots, they are flat, why aren't they 3D? The reason for this is that the vertical span is very small compared to the radius of the earth. With keyword *zScaleFun* you can specify a function for the vertical exaggeration. We chose to make this a function handle, so you can also apply a logarithmic transformation to view the deep ocean floor (4000m) and shelf depths (order 40 m) in the same plot. The default transformation is one that is appropriate for typical foreshore and dune bathymetry (MSL -20 to +30) in the Netherlands, where we first lift the surface entirely out of the water (adding deepest depth of 20m), and then apply a vertical exaggeration factor of 5: *zScaleFun*: $@(z)(z+20).*5$. For this example we will have to do something more extreme: we lift the surface somewhat out of the water, and then multiply by $1e5$ (which is roughly 10 times the thickness of the atmosphere).

```
KMLmesh(lat,lon,z,'fileName','KMLmesh_white_zscaled.kmz','lineColor',[1 1 1],'zScaleFun',@(z)(z+1).*5e4)
```



When we have judged the appropriate scaling in [KMLmesh_white_zscaled.kmz](#) by fast testing with KMLmesh, we can make a slower KMLsurf plot: [KMLsurf_zscaled.kmz](#).



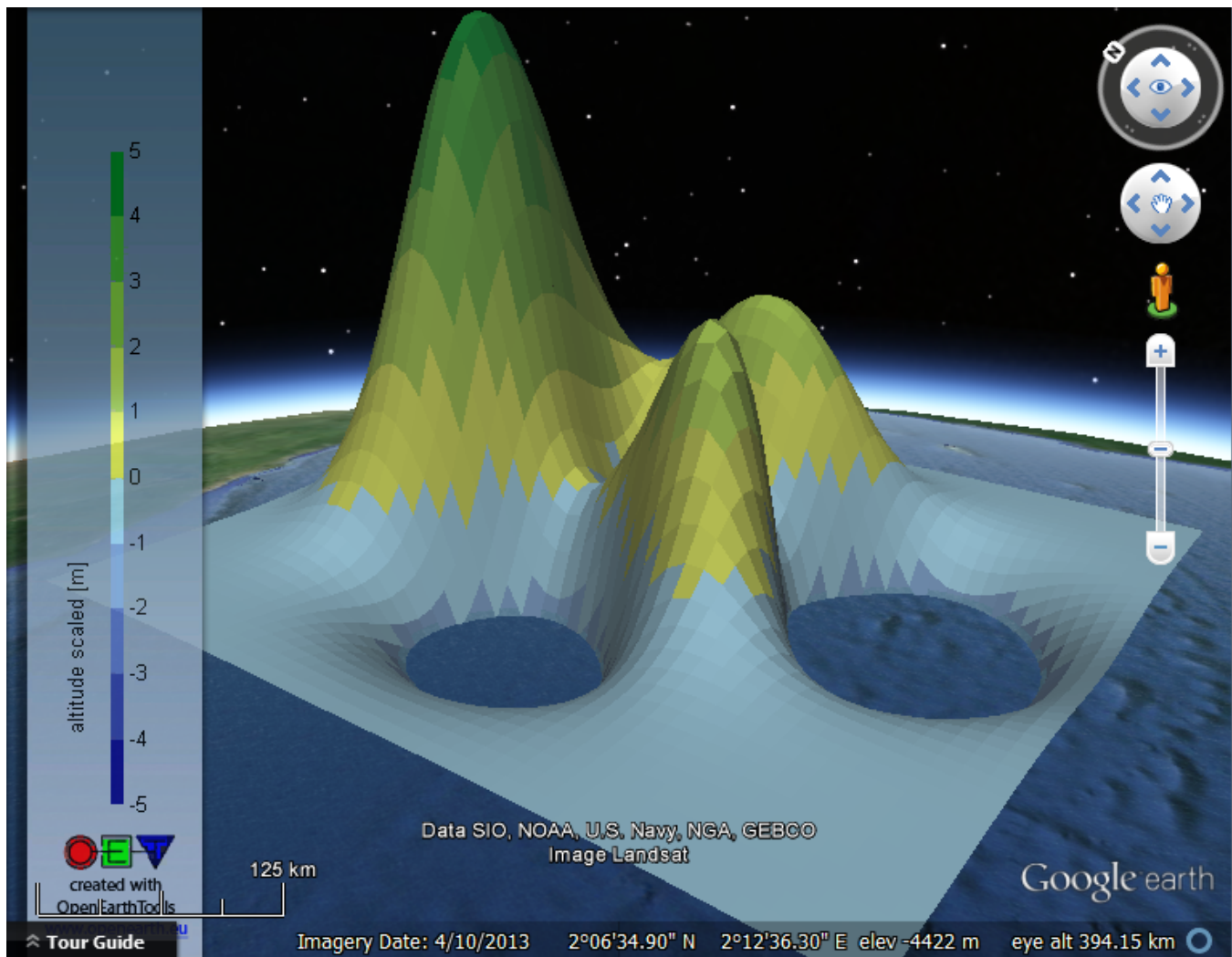
```
KMLsurf(lat,lon,z,'fileName','KMLsurf_zscaled.kmz','zScaleFun',@(z)(z+1).*5e4)
```

Adapting color: *cLim,colorMap,colorSteps,CBcolorTitle*

When you call KMLsurf without arguments, many keywords deal with colors. You can tweak these to tailor the plot for your purposes. For bathymetries, we created the symmetric `colormapbathymetry` colormap, that has blues in its first half, and yellow just above 0 (sandy beaches) and greens for the remainder. This one is optimized for color limits where MSL is exactly in the middle, here we choose `-5 . . 5`, and apply 10 `colorSteps` so each color exactly represents a range of 1 (ensure the the color ticks coincide with the color changes). To keep such long calls readable, we end each line with `...` and pad the next line with spaces to the commas line up as a column.

```
KMLsurf(lat,lon,z,'fileName','KMLsurf_zscaled_clim.kmz',...
        'zScaleFun',@(z)(z+1).*5e4,...
        'cLim',[-5 5],...
        'colorMap',@(m) colormapbathymetry(m),...
        'colorSteps',10,...
        'CBcolorTitle','altitude scaled [m]')
```

[KMLsurf_zscaled_clim.kmz](#) now has good z scaling and good color scaling.



Adding time: *TimeIn*, *TimeOut*

For movies you have to activate the inherent time functionality of the kml standard behind Google Earth. By default, objects are static, unless the properties *TimeIn* and *TimeOut* are set. Make sure the time window between *TimeIn* and *TimeOut* is sufficiently long to see the object, and make sure that the subsequent object's *TimeIn* is not after the previous' objects *TimeOut*, to prevent flickering. A small overlap can be prescribed. Easiest is to define *TimeOut* relative to *TimeIn*, by adding 1 (1 day, Matlab's standard *datetime* time unit). Google Earth always plots a later object on top of a previous object, so only in gaps previous objects are visible. In fact, you can set the Google time slides to have a time span that contains multiple time frames (drag left part of slider to left, away from right part of slider). For faster movies, make the Google Earth time slides as narrow as possible. For cycling movies, such as seasons or tides, make sure the first and last image have a smooth transition (they should not be duplicates of the same state), and enable the loop animation option to produce [KMLsurf_movie.kmz](#).

```

%% make a kml file for each movie frame
T = 12;
for it=1:T
kmlnames{it} = ['KMLsurf_movie_',num2str(it,'%0.2d'),'kml'];
[~,colorbarname]=KMLsurf(lat,lon,z*cos(2*pi*it/T),'fileName',kmlnames{it},...
    'zScaleFun',@(z)(z+1).*5e4,...
    'CBcLim',[-5 5],...
    'colorMap',@(m) colormapbathymetry(m),...
    'colorSteps',10,...
    'colorbar',it==T,...
    'timeIn',it,...
    'timeOut',it+1,...
    'CBcolorTitle','altitude scaled [m]');
end
%% merge time frames into one kml
KMLmerge_files('fileName','KMLsurf_movie.kml','sourceFiles',kmlnames)
%% merge kml with colorbar into kmz, and cleanup intermediate files
KML2kmz('KMLsurf_movie.kml',colorbarname)
deletefile('KMLsurf_movie.kml',colorbarname,kmlnames)

```

