GES393

QGIS Tutorial

Using Orthographic Projection to Map Global Connections

Marko Petrovic

Overview

Azimuth orthographic projection displays the Earth the way it looks when viewed from space. It shows a single hemisphere centered around a specified center point.

Another way to think of it is as an orthographic projection of a single hemisphere onto a plane.

As a side effect of this projection, features near the edges of the hemisphere appear squished and lines such as latitudes and longitudes that might appear straight under other projections such as UTM, turn into arcs following the curvature of the Earth.

Instructions below demonstrate how to generate an azimuth orthographic projection centered around Chicago, and map some routes to a few airports around the world.

Instructions

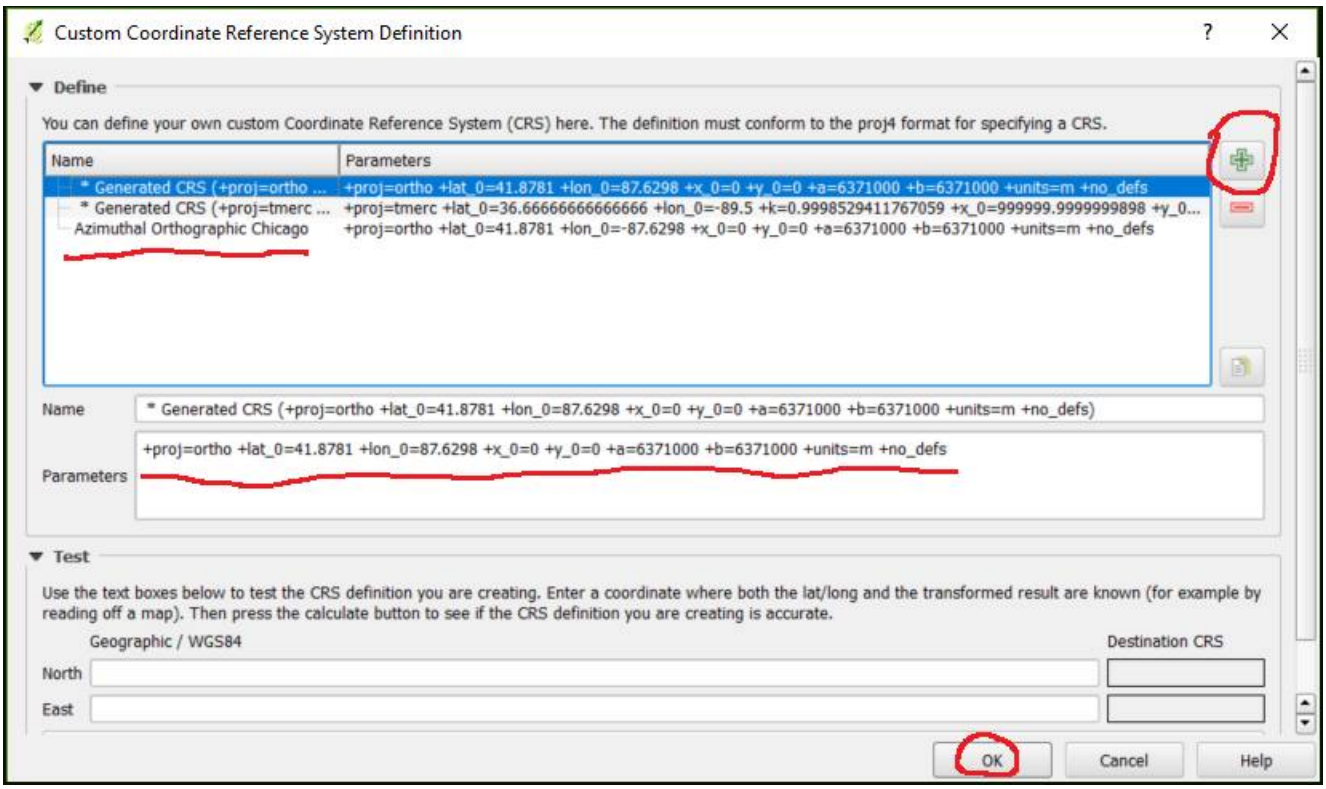
Step 1: Setting up the QGIS project.
Download a shape file with world's country boundaries. Either of the two levels of detail in the following link will do: http://thematicmapping.org/downloads/world_borders.php
Unzip the file and import it into QGIS (you can just drag the .shp file into the QGIS window).

Step 2: Defining and applying a custom projection
QGIS's default coordinate reference system is WGS 84. Let's define an azimuthal orthographic projection centered around Chicago. In the menu go into Settings > Custom CRS, press the plus sign on the right and create a new coordinate reference system.

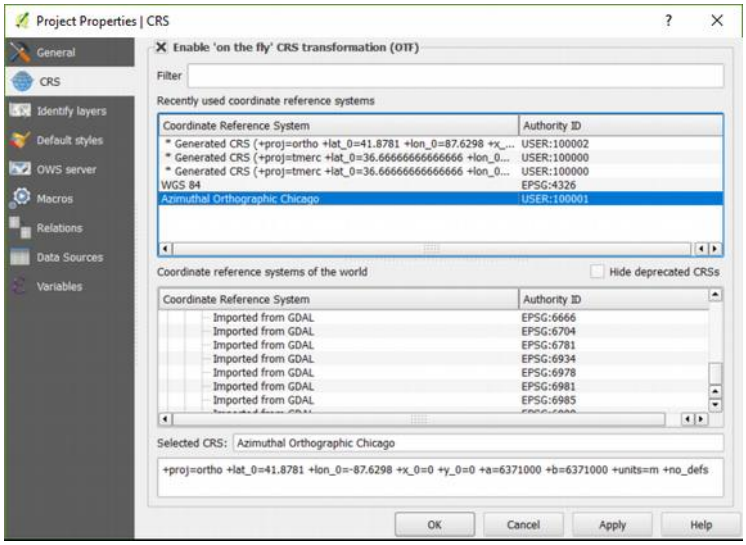
Call it something like "Azimuthal Orthographic Chicago" and under parameters put:

```
+proj=ortho +lat_0=41.8781 +lon_0=-87.6298 +x_0=0 +y_0=0 +a=6371000 +b=6371000 +units=m +no_defs
```

The underlined part defines Chicago's latitude and longitude coordinates (in decimal degrees). To center around a different point, you would change these coordinates accordingly.



We have just created the custom coordinate reference system. To apply it to the current view, go to Project > Project Properties, select it, make sure the checkbox next to *"Enable on the fly CRS transformations"* is checked, and hit apply. You should see something somilar to the screenshot on the bottom right:



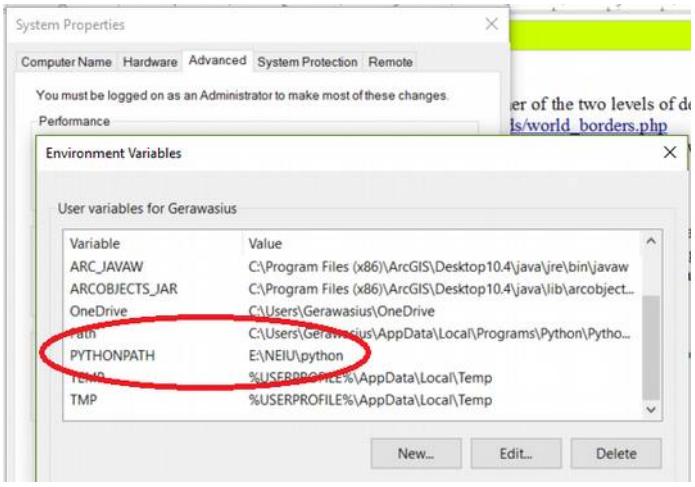
The projection looks like it really is centered around Chicago, but where did all these garbage polygons come from? Remember that only one half of the globe is visible. Projection is undefined for the other half, and therefore the position of any polygons vertices that are occluded will be undefined.

Step 3: Cleaning up the occluded geometry

There is a QGIS plugin that is supposed to do exactly what we need, but unfortunately it keeps getting stuck in mid-computation. Instead we can download a python script from this stack exchange post: https://gis.stackexchange.com/questions/78346/ortho-projection-produces-artifacts/78441

Save the script to a python file named something like `proj_clipper.py` (the extension must be .py)

In order to be able to run the script in QGIS, set the **PYTHONPATH** system variable to include the folder where you placed the file. On the right is an example of how to do it in Windows.



If you just added the PYTHONPATH, you will need to (save your project and) restart QGIS. Once you are back, open your project, and under Project Properties > CRS select WGS84 projection.

Under Plugins menu, open Python Console and type `import proj_clipper`. If you named iot differently, replace proj_clipper with the name of your python file without the extension.

In **Layers Panel** on the left select the layer you would like to clip (your country boundaries layer).
In the python console type:

```
proj_clipper.doClip(iface, 41.8781, -87.6298)
```

This will call the doClip function (which is defined in the python script). iface is the handle to the QGIS scene and the coordinates are of those Chicago. This call should clip everything that is not visible from this selection.

Unfortunately, the script only creates the clip mask (hemisphere centered around the cordinates that we passed into the doClip function, written to a layer called Clip_disk), so we will need to clip the featues manually using the **Clip** tool.

In the menu, go to Vector > Geoprocessing Tools > Clip. For input layer select the country boundaries, and for the clip layer select Clip_disc. Under clipped rite the name of the new layer. Hide everything except the clipped layer and the clip mask. The result should look like this:

Now switch back to out custom azimuthal orthographic projection. Much better! Notice that we are using the mask layer as water. Under layer properties set the styles of both layers including fill and line colors to something reasonable.

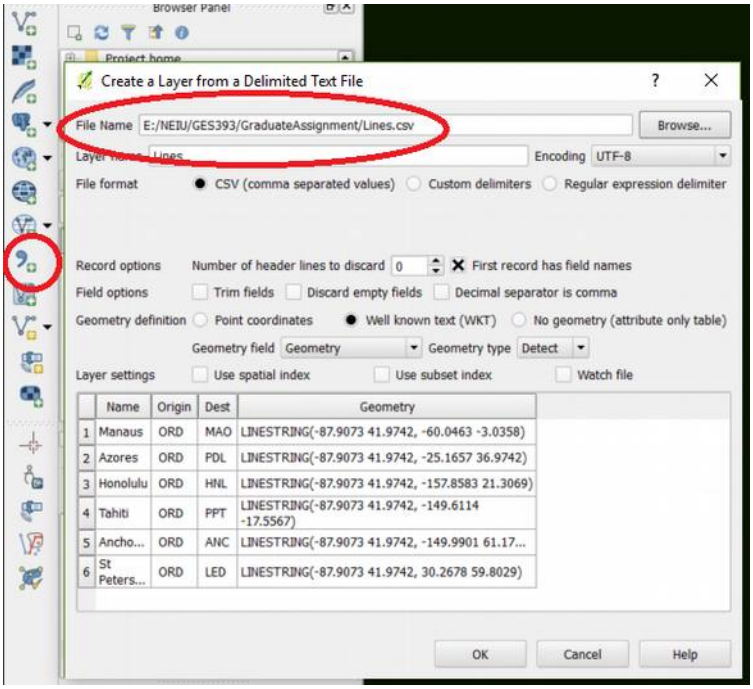
Step 4: Drawing some global routes

Let us add a couple of lines connecting different cities.
Create a csv file (name it something like Lines.csv) and add the following rows:

```
Name, Origin, Dest, Geometry
Manaus, ORD, MAO,"LINESTRING(-87.9073 41.9742, -60.0463 -3.0358)"
Azores, ORD, PDL,"LINESTRING(-87.9073 41.9742, -25.1657 36.9742)"
Honolulu, ORD, HNL,"LINESTRING(-87.9073 41.9742, -157.8583 21.3069)"
Tahiti, ORD, PPT,"LINESTRING(-87.9073 41.9742, -149.6114 -17.5567)"
Anchorage, ORD, ANC,"LINESTRING(-87.9073 41.9742, -149.9901 61.1759)"
St Petersburg, ORD, LED,"LINESTRING(-87.9073 41.9742, 30.2678 59.8029)"
```

The header row defines field names. The final cell in each row specifies a line between the two airports using WKT format.

Go back into WGS84 projection, and using the comma sign button on the left, go into "Create a layer from a delimited tet file", select your csv file. Verify that the geometry column was automatically recongnised (like in the screenshot) and press OK to import.



	Name	Origin	Dest	Geometry
1	Manaus	ORD	MAO	LINESTRING(-87.9073 41.9742, -60.0463 -3.0358)
2	Azores	ORD	PDL	LINESTRING(-87.9073 41.9742, -25.1657 36.9742)
3	Honolulu	ORD	HNL	LINESTRING(-87.9073 41.9742, -157.8583 21.3069)
4	Tahiti	ORD	PPT	LINESTRING(-87.9073 41.9742, -149.6114 -17.5567)
5	Ancho...	ORD	ANC	LINESTRING(-87.9073 41.9742, -149.9901 61.17...
6	St Peters...	ORD	LED	LINESTRING(-87.9073 41.9742, 30.2678 59.8029)

Looking at orthographic projection, the line ends are placed correctly, but the lines are straight. Why did they not get curved? The reason is, each line is represented by two points, and only the points themselves get affected by the transformation. The line connecting them is always straight.

What we need to do is subdivide each line into multiple points. Then each point will get transformed, and the segmented line connecting them all wll begin to resemble a curve. For this we will use '**Densify**' tool.

In the menu go into Vector > Geometry Tools > Densify geometries, pick the lines as the input layer, and under "vertices to add" put something like 50. The more vertices, the smoother the curved lines will be. Hit 'Run' and the lines in the newly created output layer should now appear as curved.

Go into Layer properties, and adjust the lines' Style and Labels (the "Name" column from the CSV that we imported can be used for labels).

Appendix

1. The following blogs and forum posts were very helpful in gathering this information:
https://flic.kr/p/F54ktD
http://polemic.nz/2014/11/21/nz-azimuth-orthographic/
http://www.undertheraedar.com/2014/12/simple-animations-with-qgis-long.html

2. Full python script for clipping to the specified azimuthal orthographic projection. Downloaded from:
https://gis.stackexchange.com/questions/78346/ortho-projection-produces-artifacts/78441

```
import numpy as np
from qgis.core import *
import qgis.utils
import sys, os, imp

def doClip(iface, lat=30, lon=110, filename='result.shp'):
  sourceLayer = iface.activeLayer()
  sourceCrs = sourceLayer.dataProvider().crs()
  targetProjString = "+proj=ortho +lat_0=" + str(lat) + " +lon_0=" + str(lon) + "+x_0=0 +y_0=0
+a=6370997 +b=6370997 +units=m +no_defs"
  targetCrs = QgsCoordinateReferenceSystem()
  targetCrs.createFromProj4(targetProjString)
  transformTargetToSrc = QgsCoordinateTransform(targetCrs, sourceCrs).transform

  def circlePolygon(nPoints=20, radius=6370000, center=[0,0]):
    clipdisc = QgsVectorLayer("Polygon?crs=epsg:4326", "Clip disc", "memory")
    angles = np.linspace(0, 2*np.pi, nPoints, endpoint=False)
    circlePoints = np.array([ transformTargetToSrc(QgsPoint(center[0]+np.cos(angle)*radius,
center[1]+np.sin(angle)*radius)) for angle in angles ])
    sortIdx = np.argsort(circlePoints[:,0])
    circlePoints = circlePoints[sortIdx,:]
    circlePoints = [ QgsPoint(point[0], point[1]) for point in circlePoints ]
    circlePoints.extend([QgsPoint(180,circlePoints[-1][1]), QgsPoint(180,np.sign(lat)*90), QgsPoint(-
180,np.sign(lat)*90), QgsPoint(-180,circlePoints[0][1])])
    circle = QgsFeature()
    circle.setGeometry(QgsGeometry.fromPolygon( [circlePoints] ) )
    clipdisc.dataProvider().addFeatures([circle])
    QgsMapLayerRegistry.instance().addMapLayer(clipdisc)
    return clipdisc
  auxDisc = circlePolygon(nPoints = 3600)
  ###### The clipping stuff
  ## Code taken from the fTools plugin
  vproviderA = sourceLayer.dataProvider()
  vproviderB = auxDisc.dataProvider()
  inFeatA = QgsFeature()
  inFeatB = QgsFeature()
  outFeat = QgsFeature()
  fitA = vproviderA.getFeatures()
  nElement = 0
  writer = QgsVectorFileWriter( filename, 'UTF8', vproviderA.fields(),
              vproviderA.geometryType(), vproviderA.crs() )
  index = QgsSpatialIndex()
  feat = QgsFeature()
  index = QgsSpatialIndex()
  fit = vproviderB.getFeatures()
  while fit.nextFeature( feat ):
    index.insertFeature( feat )
  while fitA.nextFeature( inFeatA ):
   nElement += 1
   geom = QgsGeometry( inFeatA.geometry() )
   atMap = inFeatA.attributes()
   intersects = index.intersects( geom.boundingBox() )
   first = True
   found = False
```

```
  if len( intersects ) > 0:
   for id in intersects:
    vproviderB.getFeatures( QgsFeatureRequest().setFilterFid( int( id ) ) ).nextFeature( inFeatB )
    tmpGeom = QgsGeometry( inFeatB.geometry() )
    if tmpGeom.intersects( geom ):
     found = True
     if first:
      outFeat.setGeometry( QgsGeometry( tmpGeom ) )
      first = False
     else:
      try:
       cur_geom = QgsGeometry( outFeat.geometry() )
       new_geom = QgsGeometry( cur_geom.combine( tmpGeom ) )
       outFeat.setGeometry( QgsGeometry( new_geom ) )
      except:
       GEOS_EXCEPT = False
       break
   if found:
    try:
     cur_geom = QgsGeometry( outFeat.geometry() )
     new_geom = QgsGeometry( geom.intersection( cur_geom ) )
     if new_geom.wkbType() == 0:
      int_com = QgsGeometry( geom.combine( cur_geom ) )
      int_sym = QgsGeometry( geom.symDifference( cur_geom ) )
      new_geom = QgsGeometry( int_com.difference( int_sym ) )
     try:
      outFeat.setGeometry( new_geom )
      outFeat.setAttributes( atMap )
      writer.addFeature( outFeat )
     except:
      FEAT_EXCEPT = False
      continue
    except:
     GEOS_EXCEPT = False
     continue
  del writer

  resultLayer = QgsVectorLayer(filename, sourceLayer.name() + " - Ortho: Lat " + str(lat) + ", Lon " +
str(lon), "ogr")
  QgsMapLayerRegistry.instance().addMapLayer(resultLayer)
```