proj.4 Documentation

Release 4.9.3

Gerald Evenden

Jul 05, 2017

CONTENTS

proj.4 is a standard UNIX filter function which converts geographic longitude and latitude coordinates into cartesian coordinates (and vice versa), and it is a C API for software developers to include coordinate transformation in their own software. proj.4 is maintained on GitHub.

Platform	Test Status and Coverage
Travis	
AppVeyor	
Coverage	

Full documentation is available as a single PDF at https://raw.githubusercontent.com/OSGeo/proj.4/gh-pages/proj4.pdf

CHAPTER

ONE

DOCUMENTATION

Download

Contents

Release Notes

- NEWS

Current Release

- **2016-09-02** proj-4.9.3.tar.gz (md5)

Past Releases

- **2015-09-13** proj-4.9.2.tar.gz
- **2015-03-04** proj-4.9.1.tar.gz

Binaries

Linux

- RedHat RPMs

- SUSE
- Debian
- pkgsrc
- Delphi

Docker

A Docker image with just Proj.4 binaries and a full compliment of grid shift files is available on *DockerHub*:

Windows

- OSGeo4W contains 32-bit and 64-bit Windows binaries, including support for many *grids*.

FAQ

Contents

Where can I find the list of projections and their arguments?

There is no simple single location to find all the required information. The !PostScript/PDF documents listed on the [http://trac.osgeo.org/proj/wiki main] PROJ.4 page under documentation are the authoritative source but projections

and options are spread over several documents in a form more related to their order of implementation than anything else.

The ‘‘‘proj’‘’ command itself can report the list of projections using the ‘‘‘-lp’‘’ option, the list of ellipsoids with the ‘‘‘-le’‘’ option, the list of units with the ‘‘‘-lu’‘’ option, and the list of built-in datums with the ‘‘‘-ld’‘’ option.

The [http://www.remotesensing.org/geotiff/proj_list/ GeoTIFF Projections Pages] include most of the common PROJ.4 projections, and a definition of the projection specific options for each.

- How do I do datum shifts between NAD27 and NAD83?

While the ‘‘‘nad2nad’‘’ program can be used in some cases, the ‘‘‘cs2cs’‘’ is now the preferred mechanism. The following example demonstrates using the default shift parameters for NAD27 to NAD83:

```
% cs2cs +proj=latlong +datum=NAD27 +to +proj=latlong +datum=NAD83 -117 30
```

producing:

```
117d0'2.901"W   30d0'0.407"N 0.000
```

In order for datum shifting to work properly the various grid shift files must be available. See below. More details are available in the [wiki:GenParms#nadgrids-GridBasedDatumAdjustments General Parameters] document.

How do I build/configure PROJ.4 to support datum shifting?

After downloading and unpacking the PROJ.4 source, also download and unpack the set of datum shift files. See *Download* for instructions how to fetch and install these files

On Windows the extra nadshift target must be used. For instance `nmake /f makefile.vc nadshift` in the `proj/src` directory.

A default build and install on Unix will normally build knowledge of the directory where the grid shift files are installed into the PROJ.4 library (usually /usr/local/share/proj). On Windows the library is normally built thinking that C:PROJNAD is the installed directory for the grid shift files. If the built in concept of the PROJ.4 data directory is incorrect, the `PROJ_LIB` environment can be defined with the correct directory.

How do I debug problems with NAD27/NAD83 datum shifting?

1. Verify that you have the binary files (eg. /usr/local/share/proj/conus) installed on your system. If not, see the previous question.
2. Try a datum shifting operation in relative isolation, such as with the cs2cs command listed above. Do you get reasonable results? If not it is likely the grid shift files aren't being found. Perhaps you need to define PROJ_LIB?
3. The cs2cs command and the underlying pj_transform() API know how to do a grid shift as part of a more complex coordinate transformation; however, it is imperative that both the source and destination coordinate system be defined with appropriate datum information. That means that implicitly or explicitly there must be a +datum= clause, a +nadgrids= clause or a +towgs84= clause. For instance `cs2cs +proj=latlong +datum=NAD27 +to +proj=latlong +ellps=WGS84` won't work because defining the output coordinate system as using the ellipse WGS84 isn't the same as defining it to use the datum WGS84 (use +datum=WGS84). If either the input or output are not identified as having a datum, the datum shifting (and ellipsoid change) step is just quietly skipped!
4. The `PROJ_DEBUG` environment can be defined (any value) to force extra output from the PROJ.4 library to stderr (the text console normally) with information on what data files are being opened and in some cases why a transformation fails.

```
export PROJ_DEBUG=ON
cs2cs ...
```

Note: PROJ_DEBUG support is not yet very mature in the PROJ.4 library.

5. The -v flag to cs2cs can be useful in establishing more detail on what parameters being used internally for a coordinate system. This will include expanding the definition of +datum clause.

How do I use EPSG coordinate system codes with PROJ.4?

There is somewhat imperfect translation between 2d geographic and projected coordinate system codes and PROJ.4 descriptions of the coordinate system available in the epsg definition file that normally lives in the proj/nad directory. If installed (it is installed by default on Unix), it is possible to use EPSG numbers like this:

```
% cs2cs -v +init=epsg:26711
# ---- From Coordinate System ----
#Universal Transverse Mercator (UTM)
#       Cyl, Sph
#       zone= south
# +init=epsg:26711 +proj=utm +zone=11 +ellps=clrk66 +datum=NAD27 +units=m
# +no_defs +nadgrids=conus,ntv1_can.dat
#--- following specified but NOT used
# +ellps=clrk66
# ---- To Coordinate System ----
#Lat/long (Geodetic)
#
# +proj=latlong +datum=NAD27 +ellps=clrk66 +nadgrids=conus,ntv1_can.dat
```

The proj/nad/epsg file can be browsed and searched in a text editor for coordinate systems. There are known to be problems with some coordinate systems, and any coordinate systems with odd axes, a non-greenwich prime meridian or other quirkyness are unlikely to work properly. Caveat Emptor!

How do I use 3 parameter and 7 parameter datum shifting

Datum shifts can be approximated with 3 and 7 parameter transformations. Their use is more fully described in the [wiki:GenParms#towgs84-DatumtransformationtoWGS84 towgs84] parameter discussion.

Does PROJ.4 work in different international numeric locales?

No. PROJ.4 makes extensive use of sprintf() and atof() internally to translate numeric values. If a locale is in effect that modifies formatting of numbers, altering the role of commas and periods in numbers, then PROJ.4 will not work. This problem is common in some European locales.

On unix-like platforms, this problem can be avoided by forcing the use of the default numeric locale by setting the LC_NUMERIC environment variable to C.

```
$ export LC_NUMERIC=C
$ proj ...
```

Note: NOTE: Per ticket #49, in PROJ 4.7.0 and later pj_init() operates with locale overridden to "C" to avoid most locale specific processing for applications using the API. Command line tools may still have issues.

Changing Ellipsoid / Why can't I convert from WGS84 to Google Earth / Virtual Globe Mercator?

The coordinate system definition for Google Earth, and Virtual Globe Mercator is as follows, which uses a sphere as the earth model for the Mercator projection.

```
+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0
     +x_0=0.0 +y_0=0 +k=1.0 +units=m +no_defs
```

But, if you do something like:

```
cs2cs +proj=latlong +datum=WGS84
    +to +proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0
                   +x_0=0.0 +y_0=0 +k=1.0 +units=m +no_defs
```

to convert between WGS84 and mercator on the sphere there will be substantial shifts in the Y mercator coordinates. This is because internally cs2cs is having to adjust the lat/long coordinates from being on the sphere to being on the WGS84 datum which has a quite differently shaped ellipsoid.

In this case, and many other cases using spherical projections, the desired approach is to actually treat the lat/long locations on the sphere as if they were on WGS84 without any adjustments when using them for converting to other coordinate systems. The solution is to "trick" PROJ.4 into applying no change to the lat/long values when going to (and through) WGS84. This can be accomplished by asking PROJ to use a null grid shift file for switching from your spherical lat/long coordinates to WGS84.

```
cs2cs +proj=latlong +datum=WGS84 \
    +to +proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 \
    +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +no_defs
```

Note the strategic addition of +nadgrids=@null to the spherical projection definition.

Similar issues apply with many other datasets distributed with projections based on a spherical earth model - such as many NASA datasets. This coordinate system is now known by the EPSG code 3857 and has in the past been known as EPSG:3785 and EPSG:900913. When using this coordinate system with GDAL/OGR it is helpful to include the +wktext so the exact proj.4 string will be preserved in the WKT representation (otherwise key parameters like *+nadgrids=@null* will be dropped):

```
+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0
           +units=m +nadgrids=@null +wktext  +no_defs
```

Why do I get different results with 4.5.0 and 4.6.0?

The default datum application behavior changed with the 4.6.0 release. PROJ.4 will now only apply a datum shift if both the source and destination coordinate system have valid datum shift information.

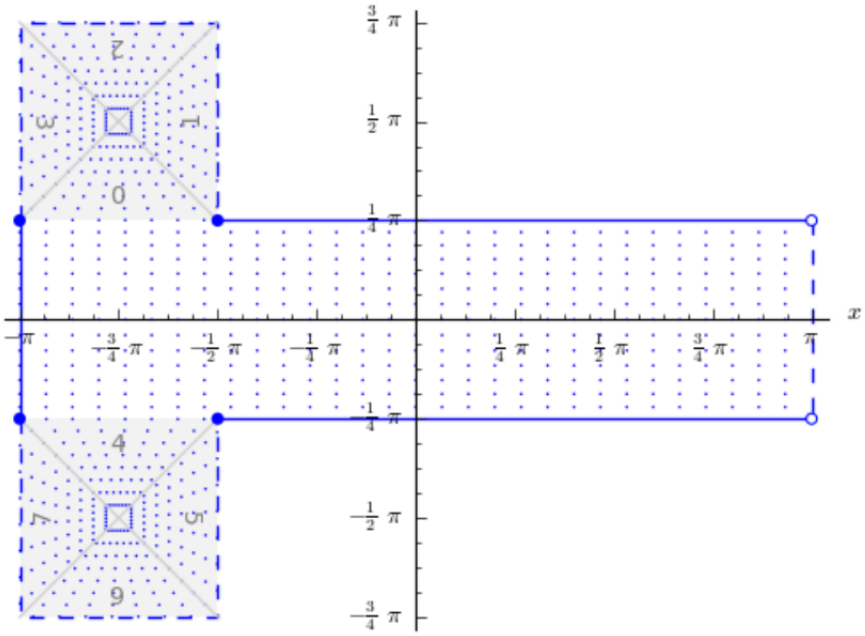
From the PROJ.4 4.6.0 Release Notes (in NEWS):

- MAJOR: Rework pj_transform() to avoid applying ellipsoid to ellipsoid transformations as a datum shift when no datum info is available.

How do I calculate distances/directions on the surface of the earth?

These are called geodesic calculations. There is a page about it here: [wiki:GeodesicCalculations]

What is the HEALPix projection and how can I use it?



The HEALPix projection is area preserving and can be used with a spherical and ellipsoidal model. It was initially developed for mapping cosmic background microwave radiation. The image below is the graphical representation of the mapping and consists of eight isomorphic triangular interrupted map graticules. The north and south contains four in which straight meridians converge polewards to a point and unequally spaced horizontal parallels. HEALPix provides a mapping in which points of equal latitude and equally spaced longitude are mapped to points of equal latitude and equally spaced longitude with the module of the polar interruptions. ||

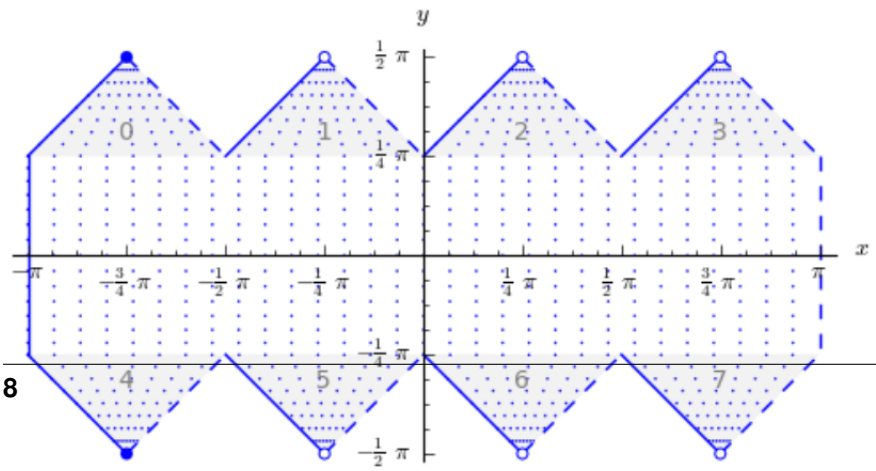
To run a forward HEALPix projection on a unit sphere model, use the following command:

```
proj +proj=healpix +lon_0=0 +a=1 -E <<EOF
0 0
EOF
```

Output of the above command.

```
0 0 0.00 0.00
```

What is the rHEALPix projection and how can I use it?



rHEALPix is a projection based on the HEALPix projection. The implementation of rHEALPix uses the HEALPix projection. The rHEALPix combines the peaks of the HEALPix into a square. The square's position can be translated and rotated across the x-axis which is a noval approach for the rHEALPix

projection. The initial intention of using rHEALPix in the Spatial Computation Engine Science Collaboration Environment (SCENZGrid).

To run a inverse rHEALPix projection on a WGS84 ellipsoidal model, use the following command:

```
proj +proj=rhealpix -f '%.2f' -I +lon_0=0 +a=1 +ellps=WGS84 +npole=0 +spole=0 -E <<EOF
0 0.7853981633974483
EOF
```

Where spole and npole are integers from the range of 0 to 3 inclusive and represent the positions of the north polar and south polar squares.

Output of above command:

```
0 0.7853981633974483 0.00 41.94
```

What options does proj.4 allow for the shape of the Earth (geodesy)?

See https://github.com/OSGeo/proj.4/blob/master/src/pj_ellps.c for possible ellipse options. For example, putting `+ellps=WGS84` uses the `WGS84` Earth shape.

What if I want a spherical Earth?

Use `+ellps=sphere`. See https://github.com/OSGeo/proj.4/blob/master/src/pj_ellps.c for the radius used in this case.

How do I change the radius of the Earth? How do I use proj.4 for work on Mars?

You can supply explicit values for the semi minor and semi major axes instead of using the symbolic "sphere" value. Eg, if the radius were 2000000m:

```
+proj=laea +lon_0=-40.000000 +lat_0=74.000000 +x_0=1000000 +y_0=1700000 +a=2000000␣
↪+b=2000000"
```

How do I do False Eastings and False Northings?

Use `+x_0` and `+y_0` in the projection string.

Applications

The proj program is limited to converting between geographic and projection coordinates within one datum.

The cs2cs program operates similarly, but allows translation between any pair of definable coor- dinate systems, including support for datum translation.

The geod program provides the ability to compute geodesic (Great Circle) computations.

proj

Proj and invproj perform respective forward and inverse transformation of cartographic data to or from cartesian data with a wide range of selectable projection functions.

Synopsis

```
proj [ -bcCeEfiIlmorsStTvVwW [ args ] ] [ +args ] file[s]
invproj [ -bcCeEfiIlmorsStTwW [ args ] ] [ +args ] file[s]
```

Description

The following control parameters can appear in any order

```
-b      Special option for binary coordinate data input and output through standard
↪input
       and standard output. Data is assumed to be in system type double floating point
       words. This option is to be used when proj is a son process and allows
↪bypassing
       formatting operations.

-i      Selects binary input only (see -b option).

-C      Check. Invoke all built in self tests and report. Get more verbose report by
       preceding with the -V option).

-I      alternate method to specify inverse projection. Redundant when used with
↪invproj.

-o      Selects binary output only (see -b option).

-ta     A specifies a character employed as the first character to denote a control
↪line
       to be passed through without  processing. This  option applicable to ascii
↪input
       only. (# is the default value).

-e string
       String is an arbitrary string to be output if an error is detected during data
       transformations. The default value is: *\t*. Note that if the -b, -i or -o
       options are employed, an error is returned as HUGE_VAL value for both return
↪values.

-E      causes the input coordinates to be copied to the output line prior to printing
↪the
       converted values.


-l[p|P|=|e|u|d]id
       List projection identifiers with -l, -lp or -lP (expanded) that can be selected
       with +proj. -l=id gives expanded description of projection id. List
       ellipsoid identifiers with -le, that can be selected with +ellps, -lu list of
       cartesian to meter conversion factors that can be selected with +units or -ld
       list of datums that can be selected with +datum.
```

```
-r      This options reverses the order of the expected input from longitude-latitude␣
↪or
        x-y to latitude-longitude or y-x.

-s      This options reverses the order of the output from x-y or longitude-latitude to
        y-x or latitude-longitude.

-S      Causes estimation of meridinal and parallel scale factors, area scale factor␣
↪and
        angular distortion, and maximum and minimum scale factors to be listed between
↪<>
        for each input point. For conformal projections meridinal and parallel scales
        factors will be equal and angular distortion zero. Equal area projections will
        have an area factor of 1.

-m mult
        The cartesian data may be scaled by the mult parameter. When processing data in
        a forward projection mode the cartesian output values are multiplied by mult
        otherwise the input cartesian values are divided by mult before inverse␣
↪projection.
        If the first two characters of mult are 1/ or 1: then the reciprocal value of␣
↪mult
        is employed.

-f format
        Format is a printf format string to control the form of the output values. For
        inverse projections, the output will be in degrees when this option is␣
↪employed.
        The  default format is "%.2f" for forward projection and DMS for inverse.

-[w|W]n
        N is the number of significant fractional digits to employ for seconds output␣
↪(when
        the  option  is  not  specified, -w3 is assumed). When -W is employed the␣
↪fields
        will be constant width and  with  leading zeroes.

-v      causes a listing of cartographic control parameters tested for and used by the
        program to be printed prior to input data. Should not be used with the -T
        option.

-V      This option causes an expanded annotated listing of the characteristics of the
        projected point. -v is implied with this option.

-T ulow,uhi,vlow,vhi,res[,umax,vmax]
        This option creates a set of bivariate Chebyshev polynomial coefficients that
        approximate the selected  cartographic projection on stdout. The values low and
        hi denote the range of the input where the u or v prefixes apply to respective
        longitude-x or latitude-y depending upon whether a forward or inverse␣
↪projection
        is selected. Res is an integer number specifying the power of 10 precision of
        the approximation. For example, a res of -3 specifies an approximation with an
        accuracy better than .001. Umax, and vmax specify maximum degree of the␣
↪polynomials
        (default: 15).
```

The `+args` run-line arguments are associated with cartographic parameters. Usage varies with projection and for a complete description consult the projection pages

Additional projection control parameters may be contained in two auxiliary control files: the first is optionally referenced with the `+init=file:id` and the second is always processed after the name of the projection has been established from either the run-line or the contents of +init file. The environment parameter `PROJ_LIB` establishes the default directory for a file reference without an absolute path. This is also used for supporting files like datum shift files.

One or more files (processed in left to right order) specify the source of data to be trans- formed. A - will specify the location of pro- cessing standard input. If no files are speci- fied, the input is assumed to be from stdin. For ASCII input data the two data values must be in the first two white space separated fields and when both input and output are ASCII all trailing portions of the input line are appended to the output line.

Input geographic data (longitude and latitude) must be in DMS format and input cartesian data must be in units consistent with the ellipsoid major axis or sphere radius units. Output geo- graphic coordinates will be in DMS (if the -w switch is not employed) and precise to 0.001" with trailing, zero-valued minute-second fields deleted.

Example

The following script

```
proj +proj=utm  +lon_0=112w  +ellps=clrk66
-r <<EOF
45d15'33.1"   111.5W
45d15.551666667N   -111d30
+45.25919444444    111d30'000w
EOF
```

will perform UTM forward projection with a stan- dard UTM central meridian nearest longitude 112W. The geographic values of this example are equivalent and meant as examples of various forms of DMS input. The x-y output data will appear as three lines of:

```
460769.27     5011648.45
```

cs2cs

`cs2cs` performs transformation between the source and destination cartographic coordinate system on a set of input points. The coordinate system transformation can include translation between projected and geographic coordinates as well as the application of datum shifts.

Synopsis

```
cs2cs [ -eEfIlrstvwW [ args ] ] [ +opts[=arg] ] [ +to [+opts[=arg]] ] file[s]
```

Description

The following control parameters can appear in any order:

```
-I     method to specify inverse translation, convert from +to coordinate system to␣
↪the
       primary coordinate system defined.

-ta    A specifies a character employed as the first character to denote a control ␣
↪line
```

```
       to  be passed through without processing. This option  applicable  to  ascii
↪input
       only. (# is the default value).

-e string
       String is an arbitrary string to be output if an error is detected  during
↪data
       transformations. The  default value is: *\t*. Note that if the -b, -i or -o
       options are employed, an error is returned as HUGE_VAL value for both
       return values.

-E     causes the input coordinates to be copied to the output line prior to printing
↪ the
       converted values.

-l[p|P|=|e|u|d]id
       List projection identifiers with -l, -lp or -lP (expanded) that can be selected
       with +proj. -l=id gives expanded description of projection id. List
       ellipsoid identifiers with -le, that can be selected with +ellps,-lu list  of
       cartesian to meter conversion factors that can be selected with +units or -ld
       list of datums that can be selected with +datum.

-r     This options reverses the order of the expected input from longitude-latitude
↪or
       x-y to latitude-longitude or y-x.

-s     This options reverses the order of the output from x-y or longitude-latitude to
       y-x or latitude-longitude.

-f format
       Format is a printf format string to control the form of the output values. For
       inverse projections, the output will be in degrees when this option is
↪employed.
       If a format is specified for inverse projection the output data will be in
↪deci-
       mal  degrees. The default format is "%.2f" for forward projection and DMS for
       inverse.

-[w|W]n
       N is the number of significant fractional digits to employ for seconds output
↪(when
       the option is not specified, -w3 is assumed). When -W is employed the fields
       will be constant width and with leading zeroes.

-v     causes a listing of cartographic control parameters tested for and used by the
       program to be printed prior to input data.
```

The +args run-line arguments are associated with cartographic parameters. Usage varies with projection and for a complete description consult the projection pages

The cs2cs program requires two coordinate system definitions. The first (or primary is defined based on all projection parameters not appearing after the +to argument. All projection parameters appearing after the +to argument are considered the definition of the second coordinate system. If there is no second coordinate system defined, a geographic coordinate system based on the datum and ellipsoid of the source coordinate system is assumed. Note that the source and destination coordinate system can both be projections, both be geographic, or one of each and may have the same or different datums.

Additional projection control parameters may be contained in two auxiliary control files: the first is optionally ref-

erenced with the `+init=file:id` and the second is always processed after the name of the projection has been established from either the run-line or the contents of `+init` file. The environment parameter PROJ_LIB establishes the default directory for a file reference without an absolute path. This is also used for supporting files like datum shift files.

One or more files (processed in left to right order) specify the source of data to be transformed. A - will specify the location of processing standard input. If no files are specified, the input is assumed to be from stdin. For input data the two data values must be in the first two white space separated fields and when both input and output are ASCII all trailing portions of the input line are appended to the output line.

Input geographic data (longitude and latitude) must be in DMS or decimal degrees format and input cartesian data must be in units consistent with the ellipsoid major axis or sphere radius units. Output geographic coordinates will normally be in DMS format (use `-f %.12f` for decimal degrees with 12 decimal places), while projected (cartesian) coordinates will be in linear (meter, feet) units.

Example

The following script

```
cs2cs +proj=latlong +datum=NAD83
      +to +proj=utm +zone=10  +datum=NAD27
-r <<EOF
45d15'33.1"   111.5W
45d15.551666667N   -111d30
+45.25919444444    111d30'000w
EOF
```

will transform the input NAD83 geographic coordinates into NAD27 coordinates in the UTM projection with zone 10 selected. The geographic values of this example are equivalent and meant as examples of various forms of DMS input. The x-y output data will appear as three lines of:

```
1402285.99      5076292.42 0.000
```

geod

`geod` (direct) and `invgeod` (inverse) perform geodesic ("Great Circle") computations for determining latitude, longitude and back azimuth of a terminus point given a initial point latitude, longitude, azimuth and distance (direct) or the forward and back azimuths and distance between an initial and terminus point latitudes and longitudes (inverse). The results are accurate to round off for $|f| < 1/50$, where f is flattening.

`invgeod` may not be available on all platforms; in this case call geod with the -I option.

Synopsis

```
geod +ellps=<ellipse> [ -afFIlptwW [ args ] ] [ +args ] file[s]

invgeod +ellps=<ellipse> [ -afFIlptwW [ args  ]  ]  [  +args  ] file[s]
```

Description

The following command-line options can appear in any order:

```
-I      Specifies that the inverse geodesic computation is to be
        performed. May be used with execution of geod as an
        alternative to invgeod execution.

-a      Latitude and longitudes of the initial and terminal
        points, forward and back azimuths and distance are  output.

-ta     A specifies a character employed as the first character
        to denote a control line to be passed through without
        processing.

-le     Gives a listing of all the ellipsoids that may be
        selected with the +ellps= option.

-lu     Gives a listing of all the units that  may  be  selected
        with the +units= option.

-[f|F] format
        Format is a printf format string to control the output
        form of the geographic coordinate values (f) or distance
        value (F). The default mode is DMS for geographic
        coordinates and "%.3f" for distance.

-[w|W]n
        N is the number of significant fractional digits to
        employ for seconds output (when the option is not
        specified, -w3 is assumed). When -W is employed the fields
        will be constant width with leading zeroes.

-p      This option causes the azimuthal values to be output as
        unsigned DMS numbers between 0 and 360 degrees. Also
        note -f.
```

The `+args` command-line options are associated with geodetic parameters for specifying the ellipsoidal or sphere to use. See `proj` documentation for full list of these parameters and controls. The options are processed in left to right order from the command line. Reentry of an option is ignored with the first occurrence assumed to be the desired value.

One or more files (processed in left to right order) specify the source of data to be transformed. A - will specify the location of processing standard input. If no files are specified, the input is assumed to be from stdin.

For direct determinations input data must be in latitude, longitude, azimuth and distance order and output will be latitude, longitude and back azimuth of the terminus point. Latitude, longitude of the initial and terminus point are input for the inverse mode and respective forward and back azimuth from the initial and terminus points are output along with the distance between the points.

Input geographic coordinates (latitude and longitude) and azimuthal data must be in decimal degrees or DMS format and input distance data must be in units consistent with the ellipsoid major axis or sphere radius units. The latitude must lie in the range [-90d,90d]. Output geographic coordinates will be in DMS (if the -f switch is not employed) to 0.001" with trailing, zero-valued minute-second fields deleted. Output distance data will be in the same units as the ellipsoid or sphere radius.

The Earth's ellipsoidal figure may be selected in the same manner as program `proj` by using `+ellps=`, `+a=`, `+es=`, etc.

Geod may also be used to determine intermediate points along either a geodesic line between two points or along an arc of specified distance from a geographic point. In both cases an initial point must be specified with `+lat_1=lat` and `+lon_1=lon` parameters and either a terminus point `+lat_2=lat` and `+lon_2=lon` or a distance and azimuth

from the initial point with `+S=distance` and `+A=azimuth` must be specified.

If points along a geodesic are to be determined then either `+n_S=integer` specifying the number of intermediate points and/or `+del_S=distance` specifying the incremental distance between points must be specified.

To determine points along an arc equidistant from the initial point both `+del_A=angle` and `+n_A=integer` must be specified which determine the respective angular increments and number of points to be determined.

Examples

The following script determines the geodesic azimuths and distance in U.S. statute miles from Boston, MA, to Portland, OR:

```
geod +ellps=clrk66 <<EOF -I +units=us-mi
42d15'N 71d07'W 45d31'N 123d41'W
EOF
```

which gives the results:

```
-66d31'50.141" 75d39'13.083" 2587.504
```

where the first two values are the azimuth from Boston to Portland, the back azimuth from Portland to Boston followed by the distance.

An example of forward geodesic use is to use the Boston loca- tion and determine Portland's location by azimuth and distance:

```
geod +ellps=clrk66 <<EOF +units=us-mi
42d15'N 71d07'W -66d31'50.141" 2587.504
EOF
```

which gives:

```
45d31'0.003"N 123d40'59.985"W 75d39'13.094"
```

Note: lack of precision in the distance value compromises the precision of the Portland location.

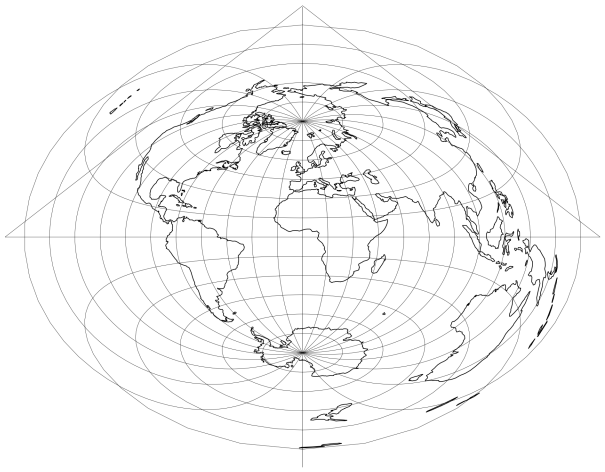
Further reading

1. GeographicLib
2. C. F. F. Karney, Algorithms for Geodesics, J. Geodesy 87, 43-55 (2013). Addendum
3. The online geodesic bibliography

Projections

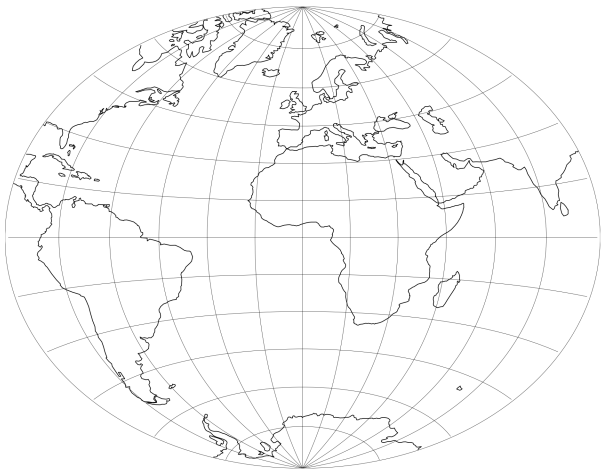
Azimuthal Equidistant

+proj=aeqd



Airy

+proj=airy

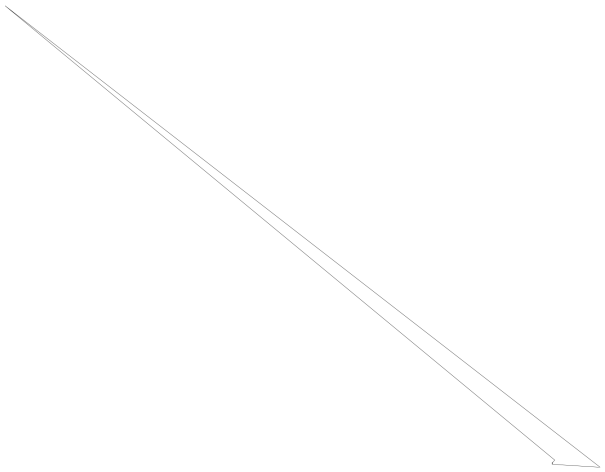


Aitoff

+proj=aitoff

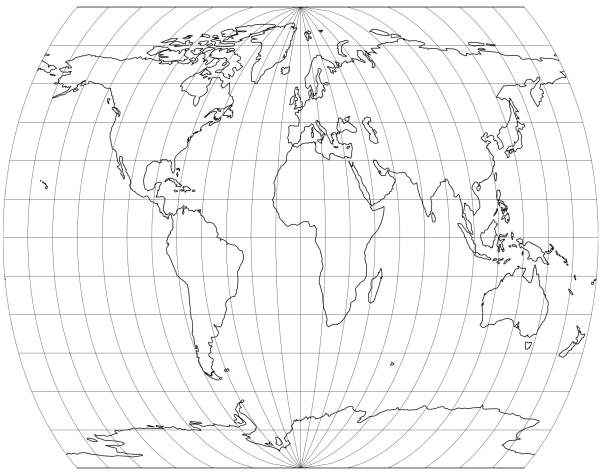
Mod. Stererographics of Alaska

+proj=alsk



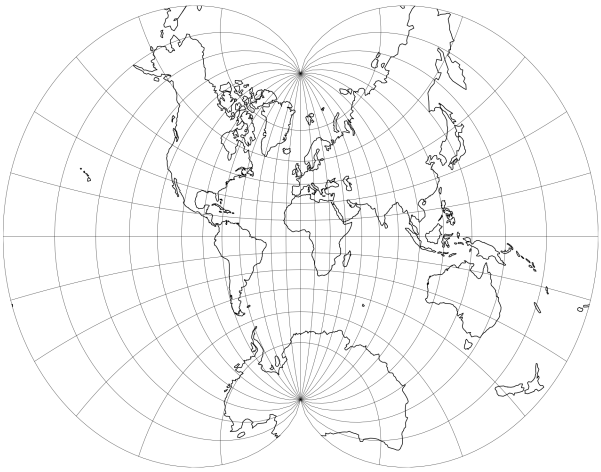
Apian Globular I

+proj=apian



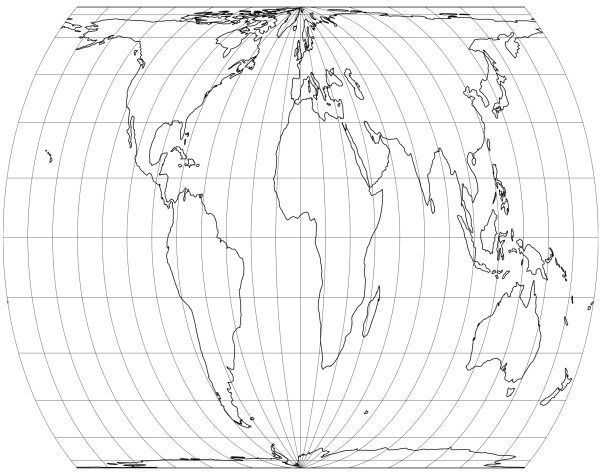
August Epicycloidal

+proj=august



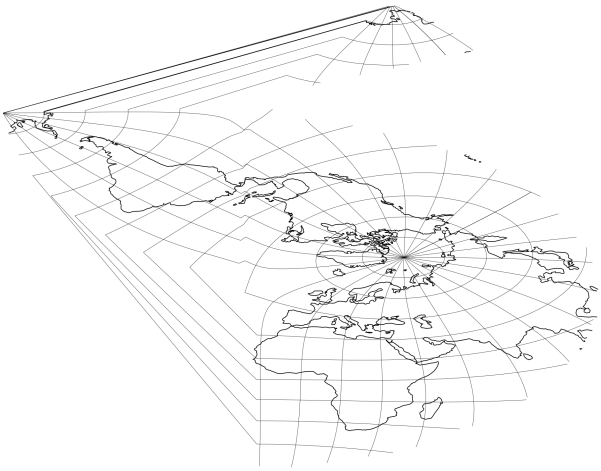
Bacon Globular

+proj=bacon



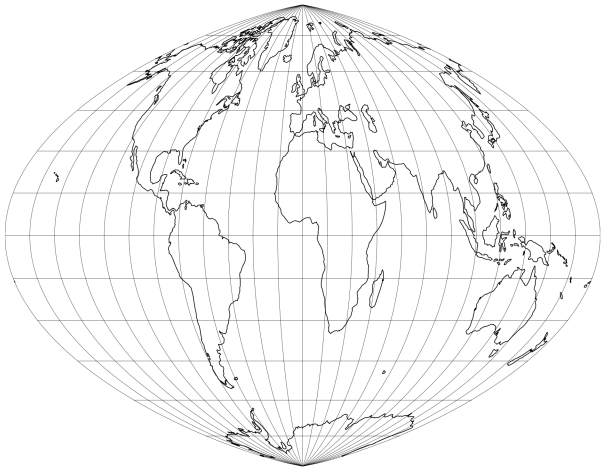
Bipolar conic of western hemisphere

+proj=bipc



Boggs Eumorphic

+proj=boggs



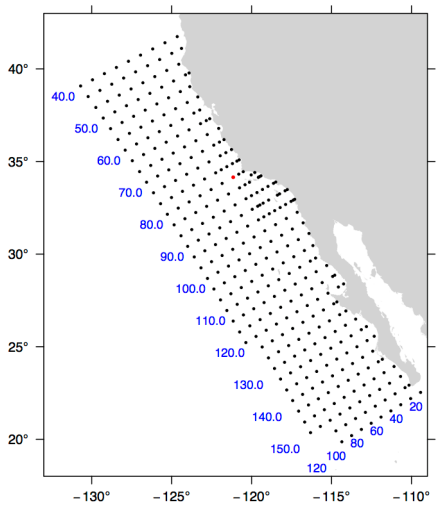
Bonne (Werner lat_1=90)

+proj=bonne +lat_1=10

Cal Coop Ocean Fish Invest Lines/Stations

The CalCOFI pseudo-projection is the line and station coordinate system of the California Cooperative Oceanic Fisheries Investigations program, known as CalCOFI, for sampling offshore of the west coast of the U.S. and Mexico.

Classification	Conformal cylindrical
Available forms	Forward and inverse, spherical and elliptical projection
Defined area	Only valid for the west coast of USA and Mexico
Implemented by	Frank Warmerdam
Options	
No special options for this projection	



The coordinate system is based on the Mercator projection with units rotated -30 degrees from the meridian so that they are oriented with the coastline of the Southern California Bight and Baja California. Lines increase from Northwest to Southeast. A unit of line is 12 nautical miles. Stations increase from inshore to offshore. A unit of station is equal to 4 nautical miles. The rotation point is located at line 80, station 60, or 34.15 degrees N, -121.15 degrees W, and is depicted by the red dot in the figure. By convention, the ellipsoid of Clarke 1866 is used to calculate CalCOFI coordinates.

The CalCOFI program is a joint research effort by the U.S. National Oceanic and Atmospheric Administration, University of California Scripps Oceanographic Institute, and California Department of Fish and Game. Surveys have been conducted for the CalCOFI program since 1951, creating one of the oldest and most scientifically valuable joint oceanographic and fisheries data sets in the world. The CalCOFI line and station coordinate system is now used by several other programs including the Investigaciones Mexicanas de la Corriente de California (IMECOCAL) program offshore of Baja California. The figure depicts some commonly sampled locations from line 40 to line 156.7 and offshore to station 120. Blue numbers indicate line (bottom) or station (left) numbers along the grid. Note that lines spaced at approximate 3-1/3 intervals are commonly sampled, e.g., lines 43.3 and 46.7.

Usage

A typical forward CalCOFI projection would be from lon/lat coordinates on the Clark 1866 ellipsoid. For example:

```
proj +proj=calcofi +ellps=clrk66 -E <<EOF
-121.15 34.15
EOF
```

Output of the above command:

```
-121.15 34.15   80.00   60.00
```

The reverse projection from line/station coordinates to lon/lat would be entered as:

```
proj +proj=calcofi +ellps=clrk66 -I -E -f "%.2f" <<EOF
80.0 60.0
EOF
```

Output of the above command:

```
80.0 60.0   -121.15 34.15
```

Mathematical definition

The algorithm used to make conversions is described in *[EberHewitt1979]* with a few corrections reported in *[WeberMoore2013]*.

Further reading

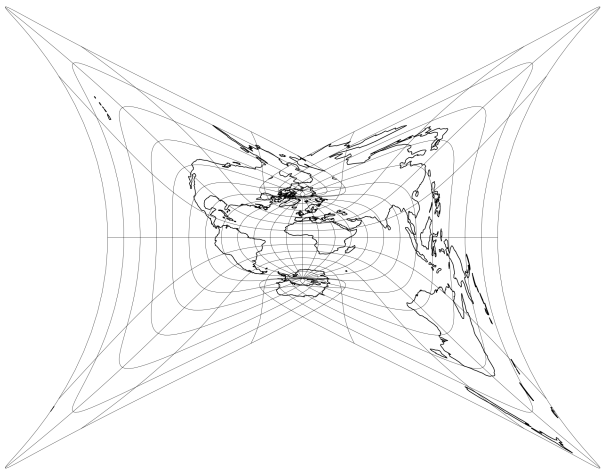
1. General information about the CalCOFI program
2. The Investigaciones Mexicanas de la Corriente de California

Cassini (Cassini-Soldner)

Although the Cassini projection has been largely replaced by the Transverse Mercator, it is still in limited use outside the United States and was one of the major topographic mapping projections until the early 20th century.

Classification	Transverse and oblique cylindrical
Available forms	Forward and inverse, Spherical and Elliptical
Defined area	Global, but best used near the central meridian with long, narrow areas
Implemented by	Gerald I. Evenden
Options	
+lat_0	Latitude of origin (Default to 0)

+proj=cass



Usage

There has been little usage of the spherical version of the Cassini, but the ellipsoidal Cassini-Soldner version was adopted by the Ordnance Survey for the official survey of Great Britain during the second half of the 19th century *[Steers1970]*. Many of these maps were prepared at a scale of 1:2,500. The Cassini-Soldner was also used for the detailed mapping of many German states during the same period.

Example using EPSG 30200 (Trinidad 1903, units in clarke's links):

```
$ echo 0.17453293 -1.08210414 | proj +proj=cass +lat_0=10.44166666666667 +lon_0=-61.
↪33333333333334 +x_0=86501.46392051999 +y_0=65379.0134283 +a=6378293.645208759␣
↪+b=6356617.987679838 +to_meter=0.201166195164 +no_defs
66644.94    82536.22
```

Example using EPSG 3068 (Soldner Berlin):

```
$ echo 13.5 52.4 | proj +proj=cass +lat_0=52.41864827777778 +lon_0=13.62720366666667␣
↪+x_0=40000 +y_0=10000 +ellps=bessel +datum=potsdam +units=m +no_defs
31343.05    7932.76
```

Mathematical definition

The formulas describing the Cassini projection are taken from Snyder's *[Snyder1987]*.

ϕ_0 is the latitude of origin that match the center of the map (default to 0). It can be set with `+lat_0`.

Spherical form

Forward projection

$$x = \arcsin(\cos(\phi)\sin(\lambda))$$

$$y = \arctan 2(\tan(\phi), \cos(\lambda)) - \phi_0$$

Inverse projection

$$\phi = \arcsin(\sin(y + \phi_0)\cos(x))$$

$$\lambda = \arctan 2(\tan(x), \cos(y + \phi_0))$$

Elliptical form

Forward projection

$$N = (1 - e^2 \sin^2(\phi))^{-1/2}$$

$$T = \tan^2(\phi)$$

$$A = \lambda \cos(\phi)$$

$$C = \frac{e^2}{1-e^2} cos^2(\phi)$$

$$x = N(A - T\frac{A^3}{6} - (8 - T + 8C)T\frac{A^5}{120})$$

$$y = M(\phi) - M(\phi_0) + N\tan(\phi)(\frac{A^2}{2} + (5 - T + 6C)\frac{A^4}{24})$$

and M() is the meridionial distance function.

Inverse projection

$$\phi' = M^{-1}(M(\phi_0) + y)$$

if $\phi' = \frac{\pi}{2}$ then $\phi = \phi'$ and $\lambda = 0$

otherwise evaluate T and N above using ϕ' and

$$R = (1 - e^2)(1 - e^2 sin^2\phi')^{-3/2}$$

$$D = x/N$$

$$\phi = \phi' - \tan\phi' \frac{N}{R}(\frac{D^2}{2} - (1 + 3T)\frac{D^4}{24})$$

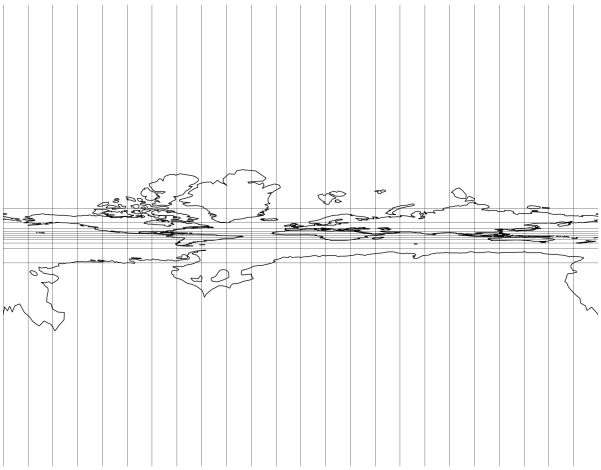
$$\lambda = \frac{(D - T\frac{D^3}{3} + (1 + 3T)T\frac{D^5}{15})}{\cos\phi'}$$

Further reading

1. Wikipedia
2. *[Snyder1987]*
3. EPSG, POSC literature pertaining to Coordinate Conversions and Transformations including Formulas

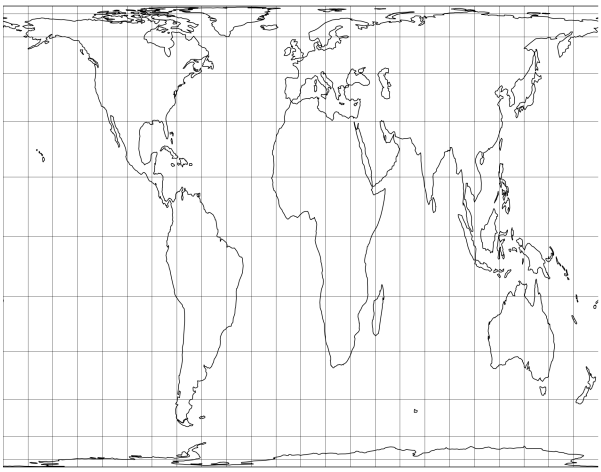
Central Cylindrical

+proj=cc



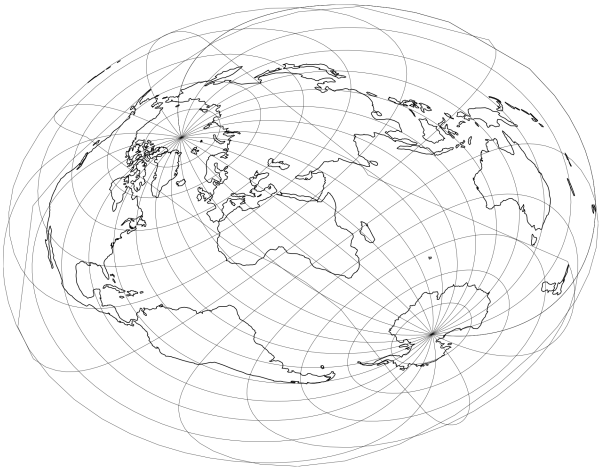
Equal Area Cylindrical

+proj=cea



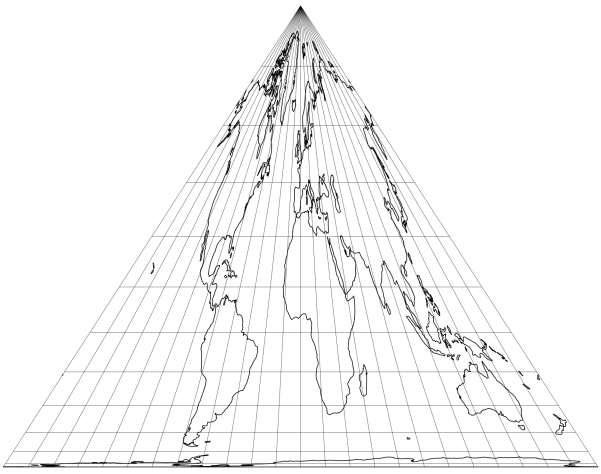
Chamberlin Trimetric

+proj=chamb +lat_1=10 +lon_1=30 +lon_2=40



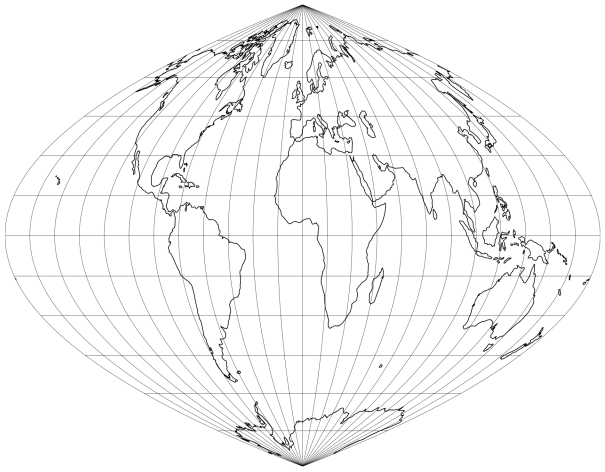
Collignon

+proj=collg



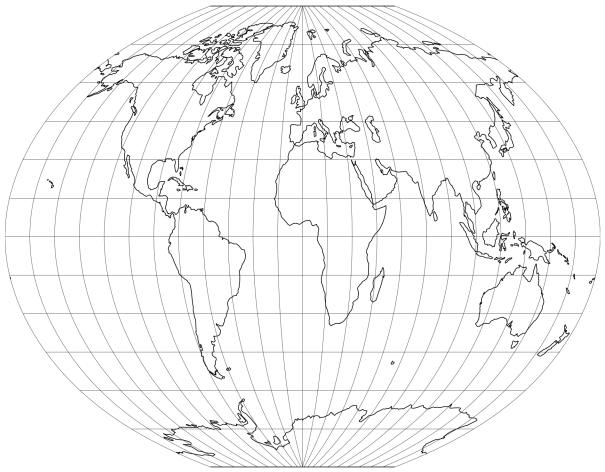
Craster Parabolic (Putnins P4)

+proj=crast

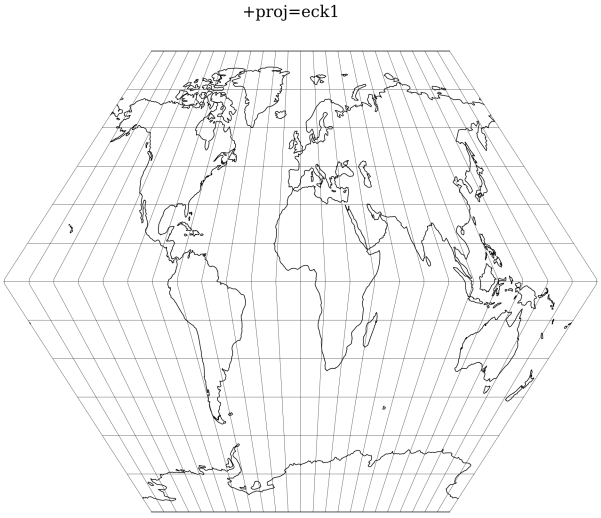


Denoyer Semi-Elliptical

+proj=denoy

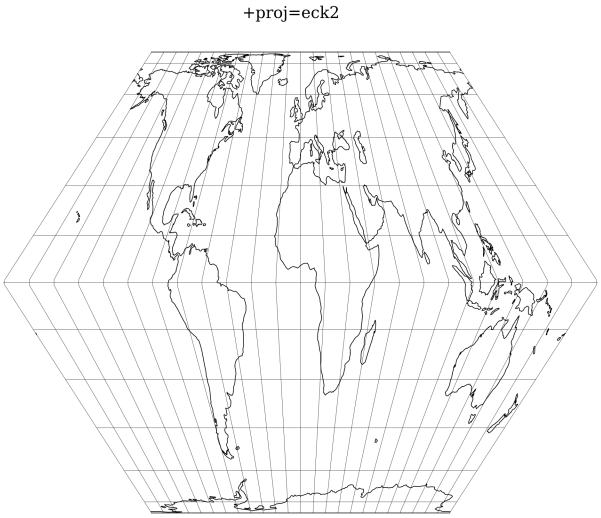


Eckert I



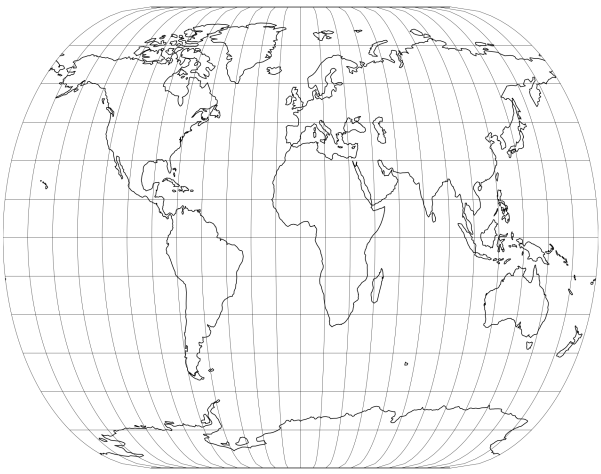
$$x = 2\sqrt{2/3\pi}\lambda(1 - |\phi|/\pi)$$
$$y = 2\sqrt{2/3\pi}\phi$$

Eckert II



Eckert III

+proj=eck3



Eckert IV

+proj=eck4

$$x = \lambda(1 + cos\phi)/\sqrt{2 + \pi}$$

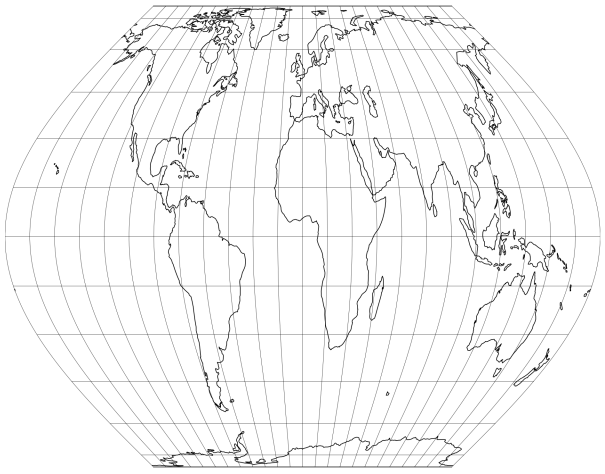
$$y = 2\phi/\sqrt{2 + \pi}$$

Eckert V

+proj=eck5

Eckert VI

+proj=eck6

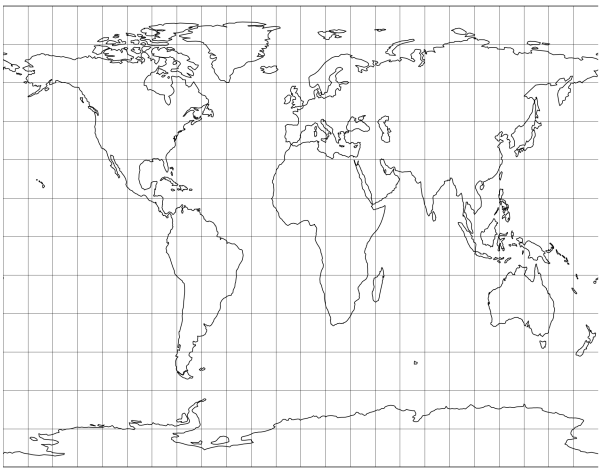


Equidistant Cylindrical (Plate Carrée)

The simplest of all projections. Standard parallels (0° when omitted) may be specified that determine latitude of true scale (k=h=1).

Classification	Conformal cylindrical
Available forms	Forward and inverse
Defined area	Global, but best used near the equator
Implemented by	Gerald I. Evenden
Options	
+lat_ts	Latitude of true scale. Defaults to 0.0
+lat_0	Center of the map : latitude of origin

+proj=eqc



Usage

Because of the distortions introduced by this projection, it has little use in navigation or cadastral mapping and finds its main use in thematic mapping. In particular, the plate carrée has become a standard for global raster datasets, such as Celestia and NASA World Wind, because of the particularly simple relationship between the position of an image pixel on the map and its corresponding geographic location on Earth.

The following table gives special cases of the cylindrical equidistant projection.

Projection Name	(lat ts=) ϕ_0
Plain/Plane Chart	0°
Simple Cylindrical	0°
Plate Carrée	0°
Ronald Miller—minimum overall scale distortion	37°30′
E.Grafarend and A.Niermann	42°
Ronald Miller—minimum continental scale distortion	43°30′
Gall Isographic	45°
Ronald Miller Equirectangular	50°30′
E.Grafarend and A.Niermann minimum linear distortion	61°7′

Example using EPSG 32662 (WGS84 Plate Carrée):

```
$ echo 2 47 | proj +proj=eqc +lat_ts=0 +lat_0=0 +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84␣
↪+datum=WGS84 +units=m +no_defs
222638.98       5232016.07
```

Example using Plate Carrée projection with true scale at latitude 30° and central meridian 90°W:

```
$ echo -88 30 | proj +proj=eqc +lat_ts=30 +lat_0=90w
-8483684.61     13358338.90
```

Mathematical definition

The formulas describing the Equidistant Cylindrical projection are all taken from Snyder's *[Snyder1987]*.

ϕ_{ts} is the latitude of true scale, that mean the standard parallels where the scale of the projection is true. It can be set with `+lat_ts`.

ϕ_0 is the latitude of origin that match the center of the map. It can be set with `+lat_0`.

Forward projection

$$x = \lambda \cos \phi_{ts}$$

$$y = \phi - \phi_0$$

Inverse projection

$$\lambda = x/cos\phi_{ts}$$

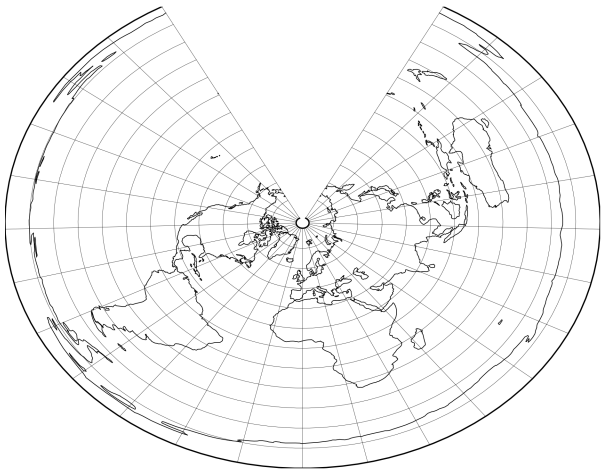
$$\phi = y + \phi_0$$

Further reading

1. Wikipedia
2. Wolfram Mathworld

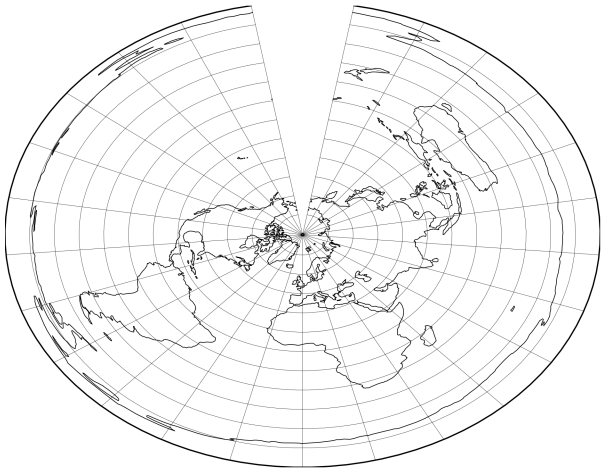
Equidistant Conic

+proj=eqdc +lat_1=55 +lat_2=60



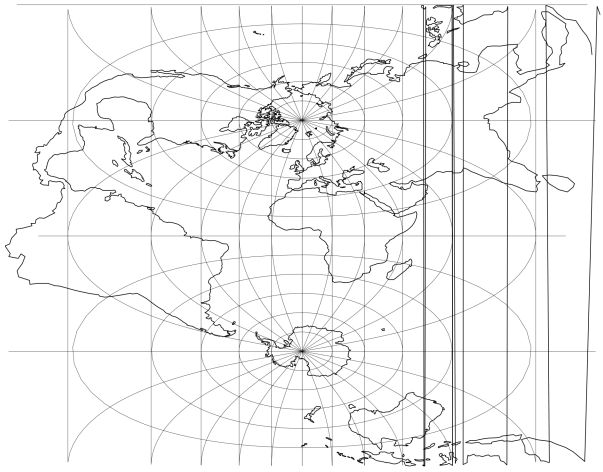
Euler

+proj=euler +lat_1=67 +lat_2=75



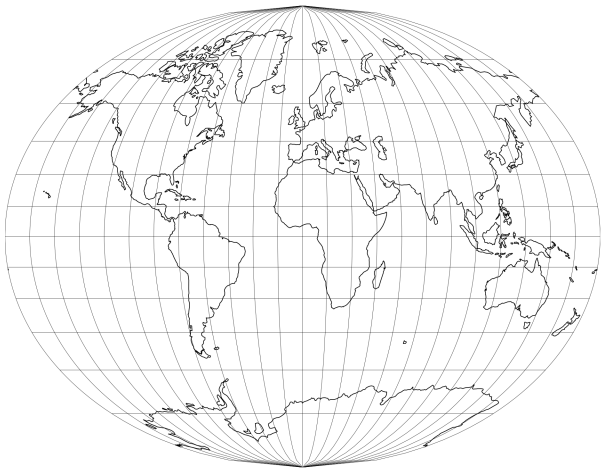
Extended Transverse Mercator

+proj=etmerc

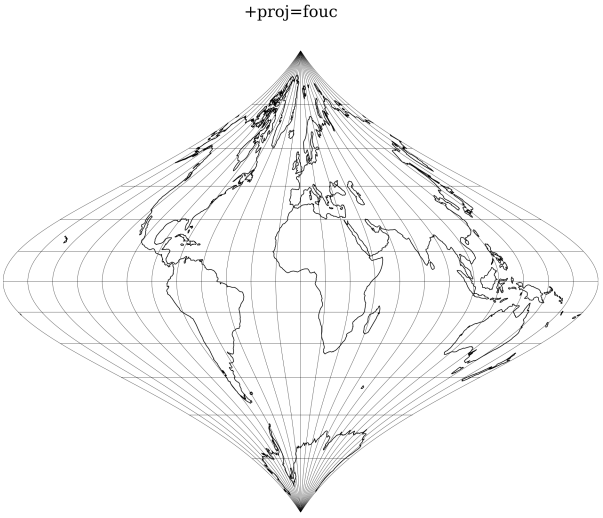


Fahey

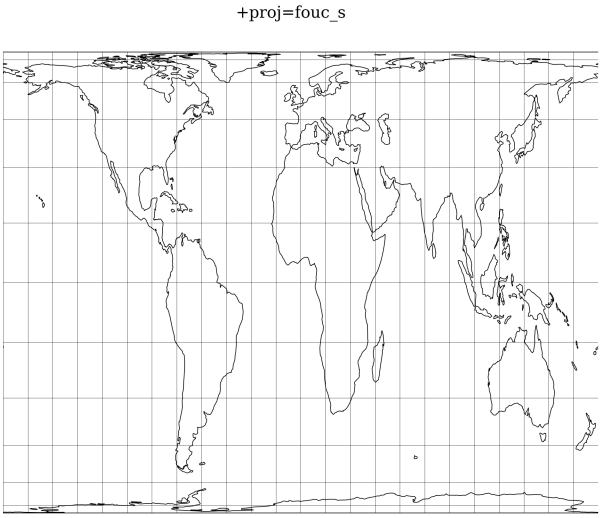
+proj=fahey



Foucaut



Foucaut Sinusoidal



The y-axis is based upon a weighted mean of the cylindrical equal-area and the sinusoidal projections. Parameter $n = n$ is the weighting factor where $0 <= n <= 1$.

$$x = \lambda \cos \phi / (n + (1 - n)\ cos\phi)$$
$$y = n\phi + (1 - n) \sin \phi$$

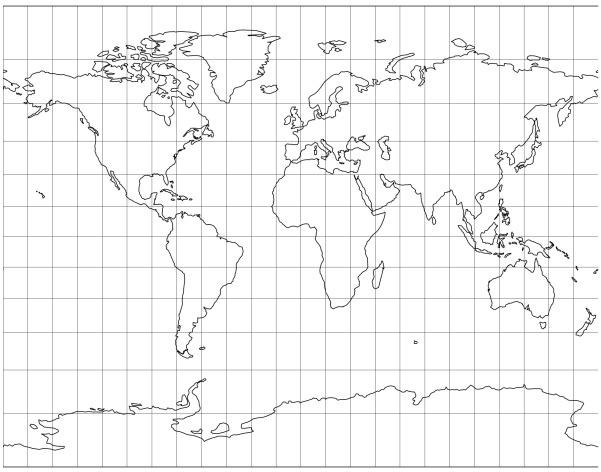
For the inverse, the Newton-Raphson method can be used to determine ϕ from the equation for y above. As $n \to 0$ and $\phi \to \pi/2$, convergence is slow but for $n = 0$, $\phi = \sin^1 y$

Gall (Gall Stereographic)

The Gall stereographic projection, presented by James Gall in 1855, is a cylindrical projection. It is neither equal-area nor conformal but instead tries to balance the distortion inherent in any projection.

Classification	Transverse and oblique cylindrical
Available forms	Forward and inverse, Spherical
Defined area	Global
Implemented by	Gerald I. Evenden
Options	No special options for this projection

+proj=gall



Usage

The need for a world map which avoids some of the scale exaggeration of the Mercator projection has led to some commonly used cylindrical modifications, as well as to other modifications which are not cylindrical. The earliest common cylindrical example was developed by James Gall of Edinburgh about 1855 (Gall, 1885, p. 119-123). His meridians are equally spaced, but the parallels are spaced at increasing intervals away from the Equator. The parallels of latitude are actually projected onto a cylinder wrapped about the sphere, but cutting it at lats. 45° N. and S., the point of perspective being a point on the Equator opposite the meridian being projected. It is used in several British atlases, but seldom in the United States. The Gall projection is neither conformal nor equal-area, but has a blend of various features. Unlike the Mercator, the Gall shows the poles as lines running across the top and bottom of the map.

Example using Gall Stereographical

```
$ echo 9 51 | proj +proj=gall +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84␣
↪+units=m +no_defs
708432.90   5193386.36
```

Example using Gall Stereographical (Central meridian 90°W)

```
$ echo 9 51 | proj +proj=gall +lon_0=90w +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84␣
↪+units=m +no_defs
7792761.91  5193386.36
```

Mathematical definition

The formulas describing the Gall Stereographical are all taken from Snyder's *[Snyder1993]*.

Spherical form

Forward projection

$$x = \frac{\lambda}{\sqrt{2}}$$

$$y = (1 + \frac{\sqrt{2}}{2}) \tan(\phi/2)$$

Inverse projection

$$\phi = 2 \arctan(\frac{y}{1 + \frac{\sqrt{2}}{2}})$$

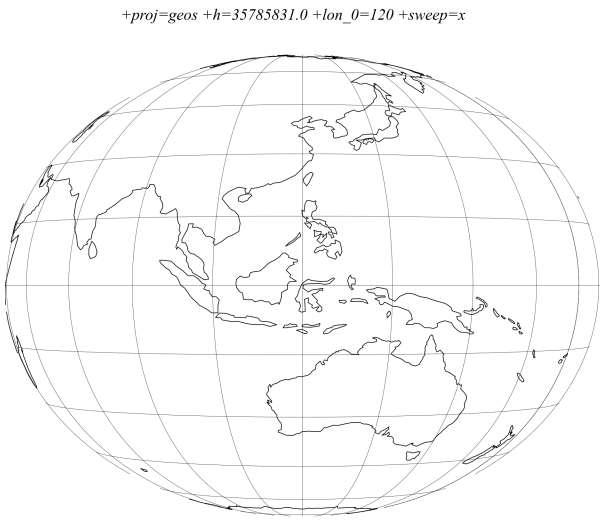
$$\lambda = \sqrt{2}x$$

Further reading

1. Wikipedia
2. Cartographic Projection Procedures for the UNIX Environment-A User's Manual

Geostationary Satellite View

Classification	Azimuthal
Available forms	Forward and inverse, spherical and elliptical projection
Defined area	Global
Implemented by	Gerald I. Evenden and Martin Raspaud$
Options	
+h	Satellite height above earth. Required.
+sweep	Sweep angle axis of the viewing instrument. Valid options are `x` and `y`. Defaults to `y`.
+lon_0	Subsatellite longitude point.



The geos projection pictures how a geostationary satellite scans the earth at regular scanning angle intervals.

Usage

In order to project using the geos projection you can do the following:

```
proj +proj=geos +h=35785831.0
```

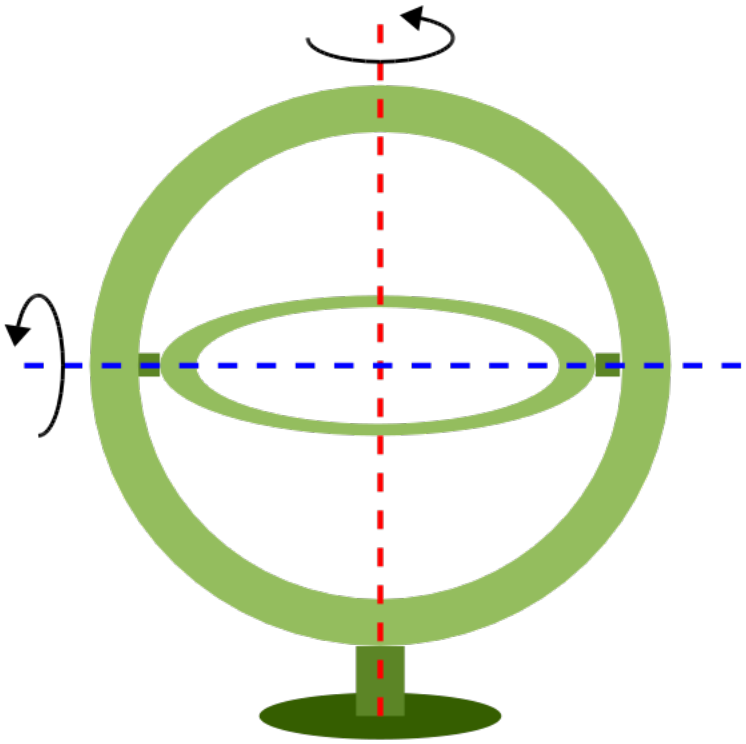
The required argument `h` is the viewing point (satellite position) height above the earth.

The projection coordinate relate to the scanning angle by the following simple relation:

```
scanning_angle (radians) = projection_coordinate / h
```

Note on sweep angle

The viewing instrument on-board geostationary satellites described by this projection have a two-axis gimbal viewing geometry. This means that the different scanning positions are obtained by rotating the gimbal along a N/S axis (or `y`) and a E/W axis (or `x`).



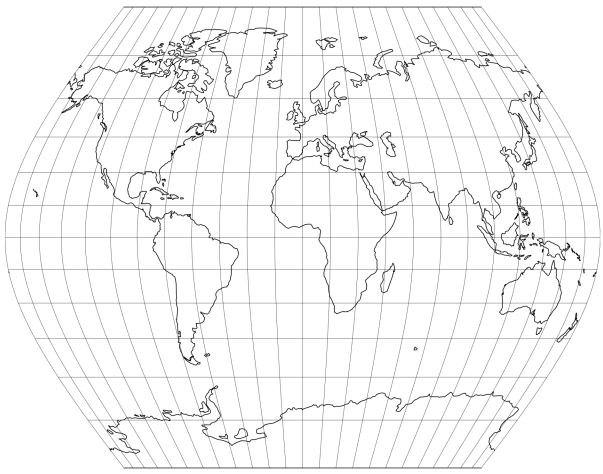
In the image above, the outer-gimbal axis, or sweep-angle axis, is the N/S axis (`y`) while the inner-gimbal axis, or fixed-angle axis, is the E/W axis (`x`).

This example represents the scanning geometry of the Meteosat series satellite. However, the GOES satellite series use the opposite scanning geometry, with the E/W axis (`x`) as the sweep-angle axis, and the N/S (`y`) as the fixed-angle axis.

The sweep argument is used to tell proj.4 which on which axis the outer-gimbal is rotating. The possible values are x or y, y being the default. Thus, the scanning geometry of the Meteosat series satellite should take sweep as x, and GOES should take sweep as y.

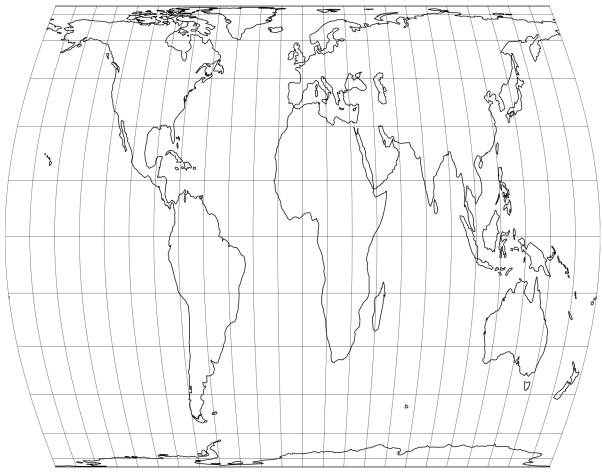
Ginsburg VIII (TsNIIGAiK)

+proj=gins8



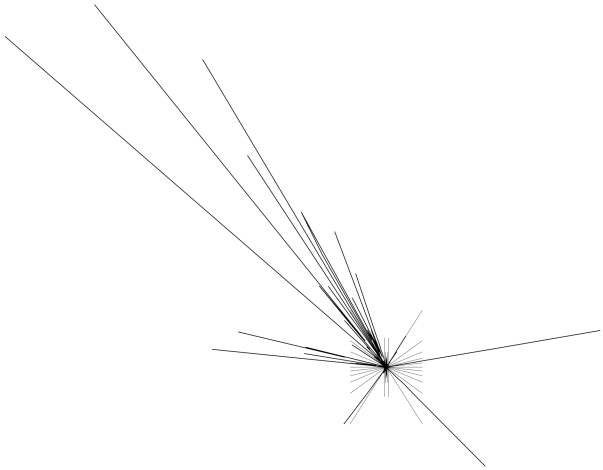
General Sinusoidal Series

+proj=gn_sinu +m=2 +n=3



Gnomonic

+proj=gnom

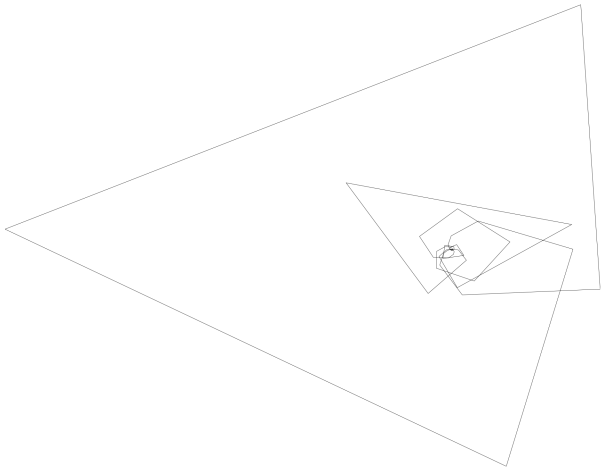


Goode Homolosine

+proj=goode

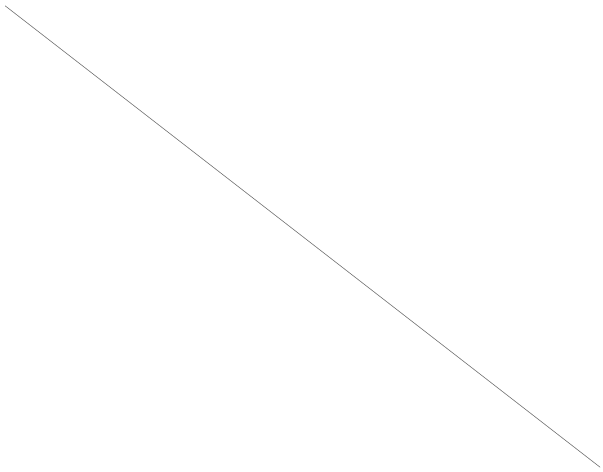
Mod. Stererographics of 48 U.S.

+proj=gs48



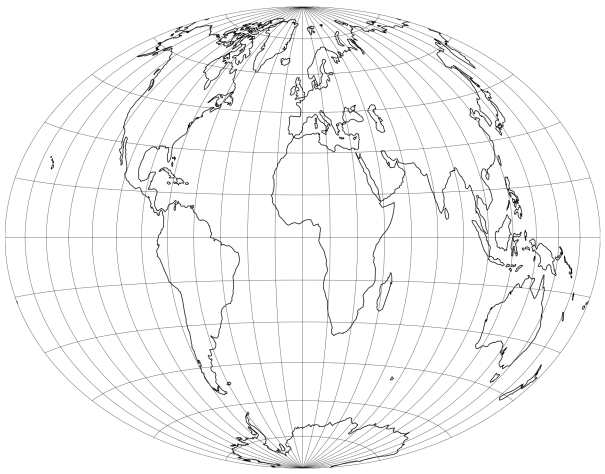
Mod. Stererographics of 50 U.S.

+proj=gs50



Hammer & Eckert-Greifendorff

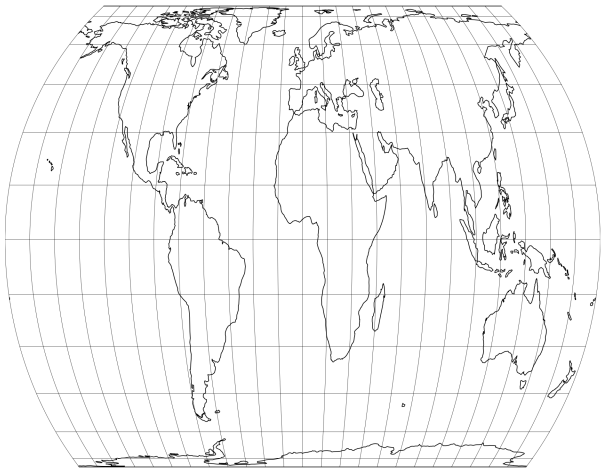
+proj=hammer



Hatano Asymmetrical Equal Area

Classification	*Pseudocylindrical Projection*
Available forms	Forward and inverse, spherical projection
Defined area	Global, but best between standard parallels
Implemented by	Gerald I. Evenden
Options	
+*lat_1*	Standard Parallel 1
+*lat_2*	Standard Parallel 2
+*sym*	Symmetric form used instead of asymmetric

+proj=hatano



Mathematical Definition

Forward

$$
\begin{aligned}
x &= 0.85\lambda \cos\theta \\
y &= C_y \sin\theta \\
P(\theta) &= 2\theta + \sin 2\theta - C_p \sin\phi \\
P'(\theta) &= 2(1 + \cos 2\theta) \\
\theta_0 &= 2\phi
\end{aligned}
$$

Condition	C_p	C_p
if `+sym` or $\phi > 0$	1.75859	2.67595
if not `+sym` and $\phi < 0$	1.93052	2.43763

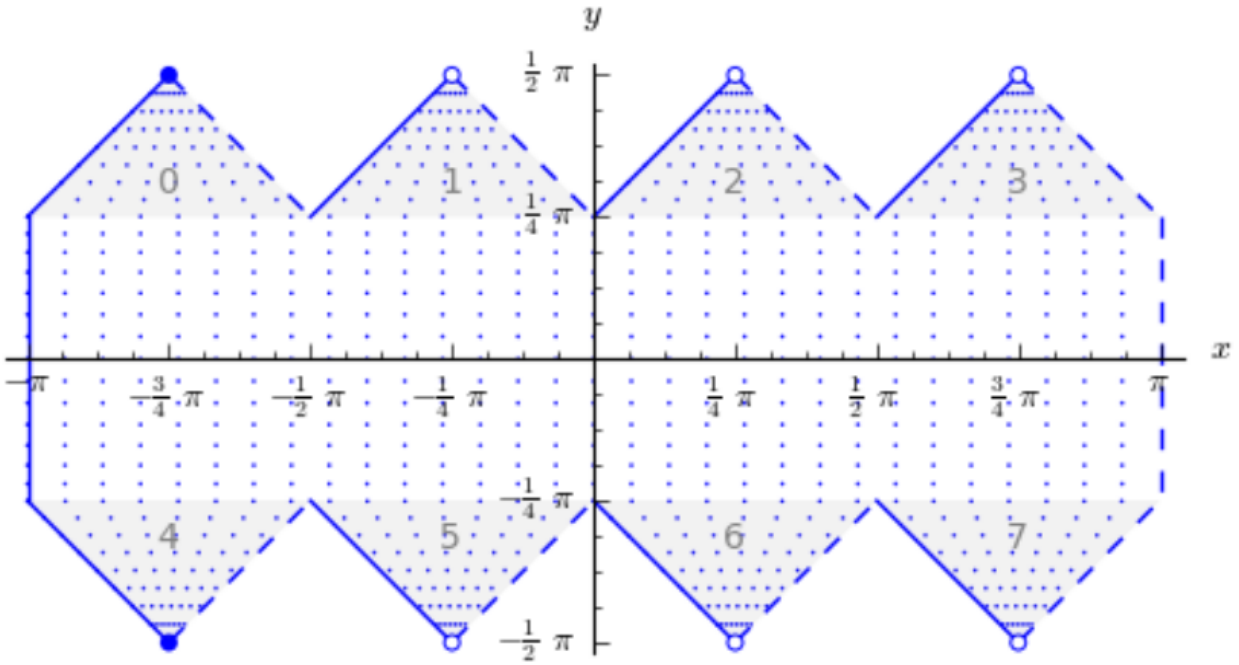
For $\phi = 0$, $y \leftarrow 0$, and $x \leftarrow 0.85\lambda$.

Further reading

1. Compare Map Projections
2. Mathworks

HEALPix

Classification	Mixed
Available forms	Forward and inverse, spherical and elliptical projection
Defined area	Global
Implemented by	Alex Raichev and Michael Speth
Options	
No special options for this projection	



The HEALPix projection is area preserving and can be used with a spherical and ellipsoidal model. It was initially developed for mapping cosmic background microwave radiation. The image below is the graphical representation of the mapping and consists of eight isomorphic triangular interrupted map graticules. The north and south contains four in which straight meridians converge polewards to a point and unequally spaced horizontal parallels. HEALPix provides a mapping in which points of equal latitude and equally spaced longitude are mapped to points of equal latitude and equally spaced longitude with the module of the polar interruptions.

Usage

To run a forward HEALPix projection on a unit sphere model, use the following command:

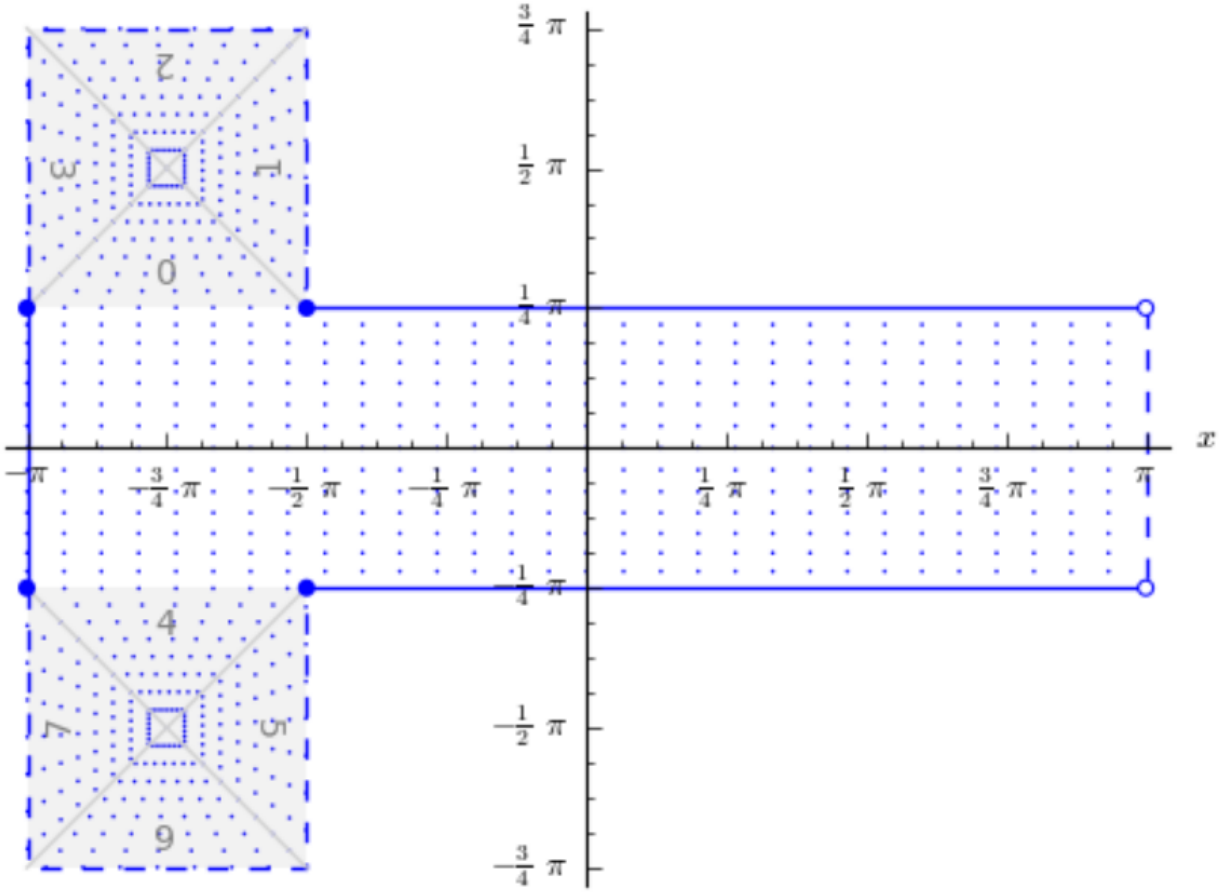
```
proj +proj=healpix +lon_0=0 +a=1 -E <<EOF
0 0
EOF
# output
0 0 0.00 0.00
```

Further reading

1. NASA
2. Wikipedia

rHEALPix

Classification	Mixed
Available forms	Forward and inverse, spherical and elliptical projection
Defined area	Global
Implemented by	Alex Raichev and Michael Speth
Options	
+north_square	Position of the north polar square. Valid inputs are 0–3. Defaults to 0.
+south_square	Position of the south polar square. Valid inputs are 0–3. Defaults to 0.



rHEALPix is a projection based on the HEALPix projection. The implementation of rHEALPix uses the HEALPix projection. The rHEALPix combines the peaks of the HEALPix into a square. The square's position can be translated and rotated across the x-axis which is a noval approach for the rHEALPix projection. The initial intention of using rHEALPix in the Spatial Computation Engine Science Collaboration Environment (SCENZGrid).

Usage

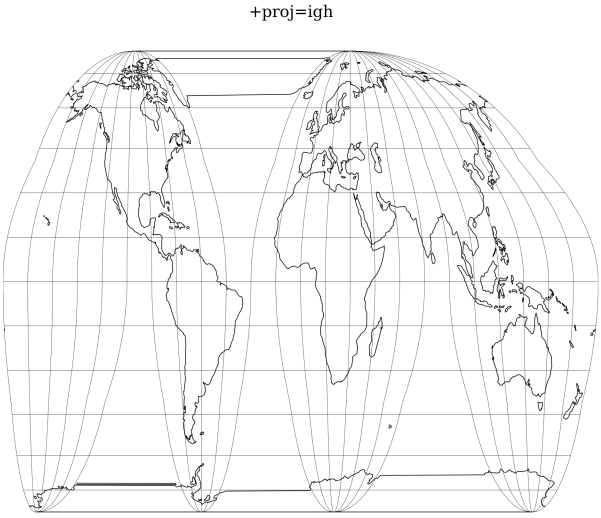
To run a rHEALPix projection on a WGS84 ellipsoidal model, use the following command:

```
proj +proj=rhealpix -f '%.2f' +ellps=WGS84 +south_square=0 +north_square=2  -E << EOF
> 55 12
> EOF
55 12   6115727.86  1553840.13
```

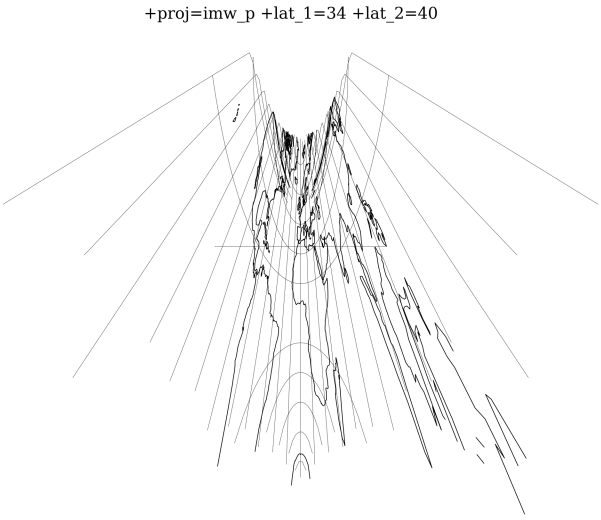
Further reading

1. NASA
2. Wikipedia

Interrupted Goode Homolosine

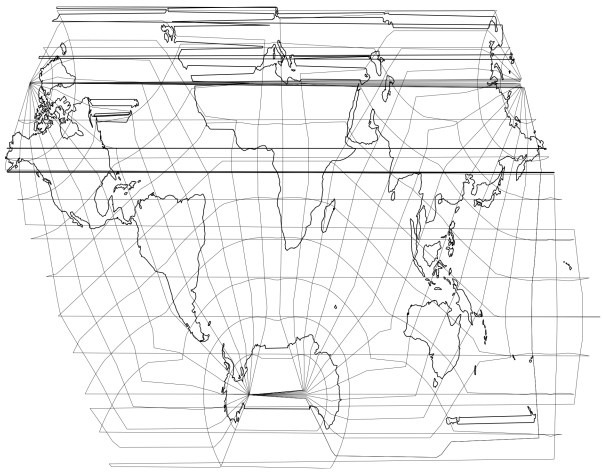


International Map of the World Polyconic



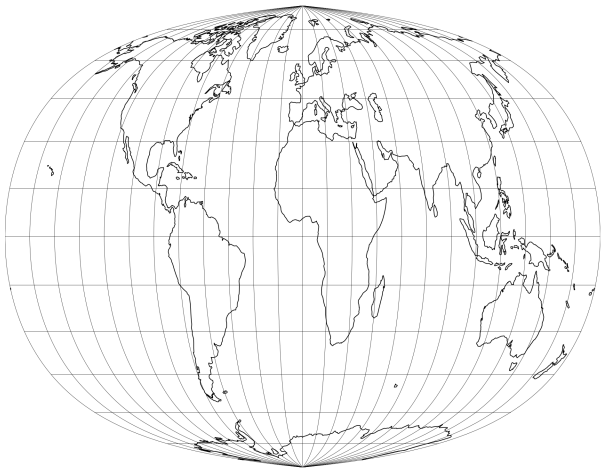
Icosahedral Snyder Equal Area

+proj=isea



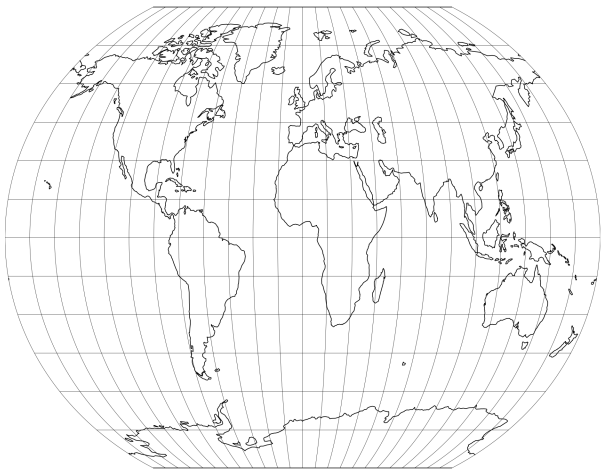
Kavraisky V

+proj=kav5



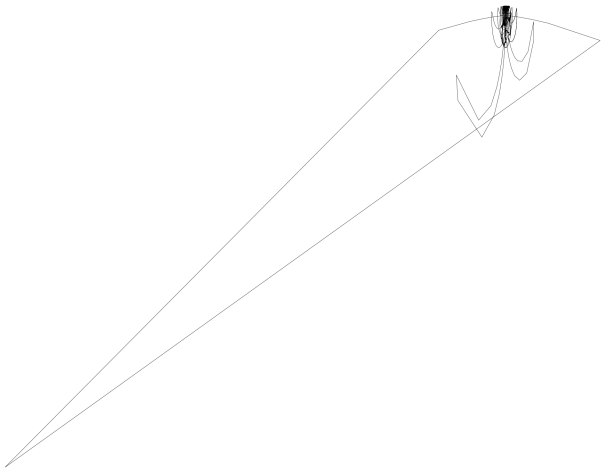
Kavraisky VII

+proj=kav7



Krovak

+proj=krovak

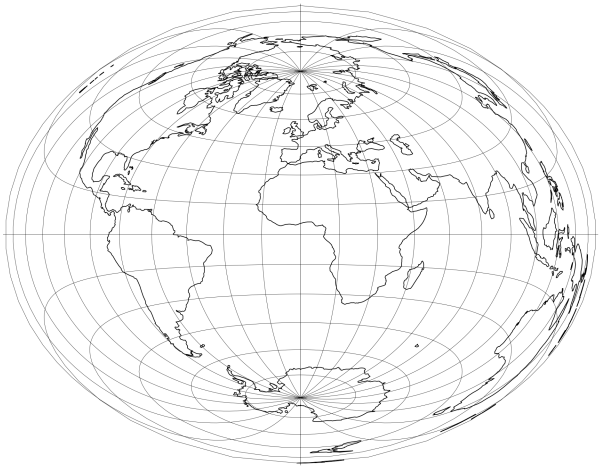


Laborde

+proj=labrd

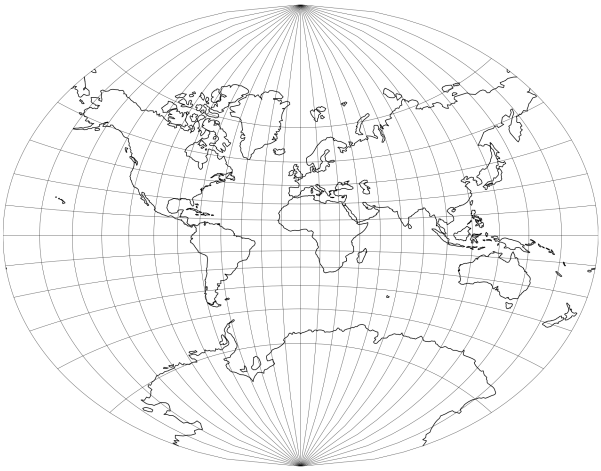
Lambert Azimuthal Equal Area

+proj=laea



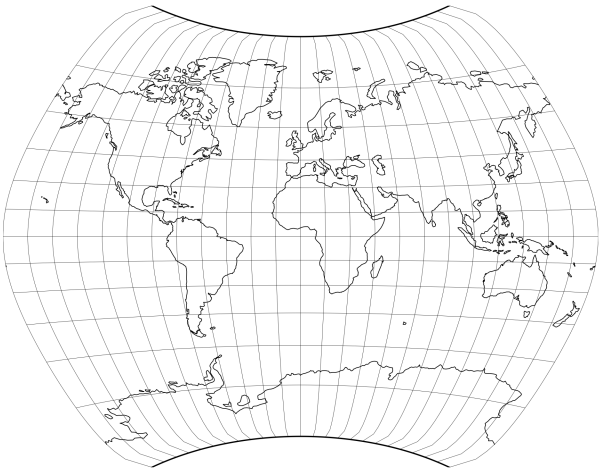
Lagrange

+proj=lagrng



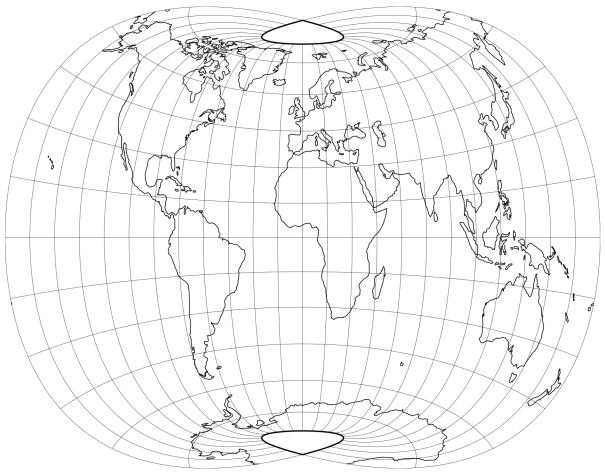
Larrivee

+proj=larr



Laskowski

+proj=lask



Lat/long (Geodetic)

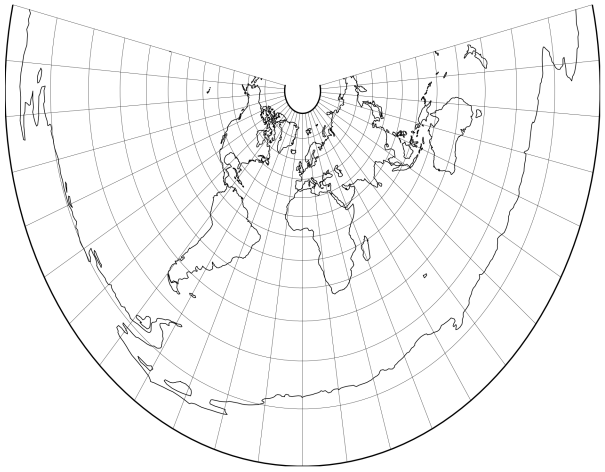
Lat/long (Geodetic alias)

Lambert Conformal Conic

+proj=lcc

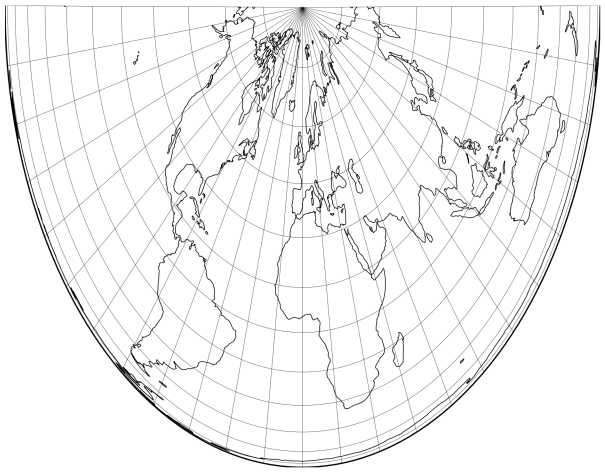
Lambert Conformal Conic Alternative

+proj=lcca +lat_0=35



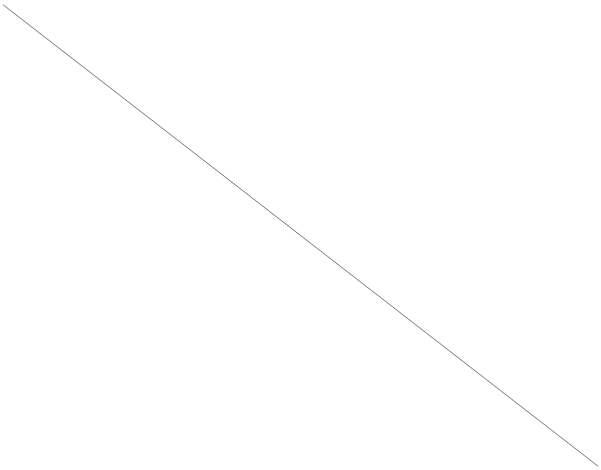
Lambert Equal Area Conic

+proj=leac



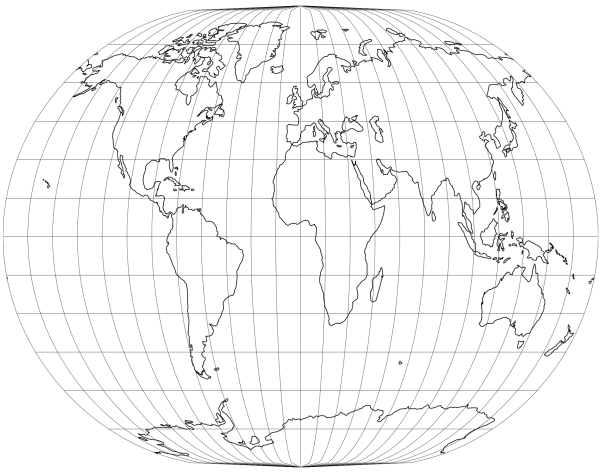
Lee Oblated Stereographic

+proj=lee_os



Loximuthal

+proj=loxim

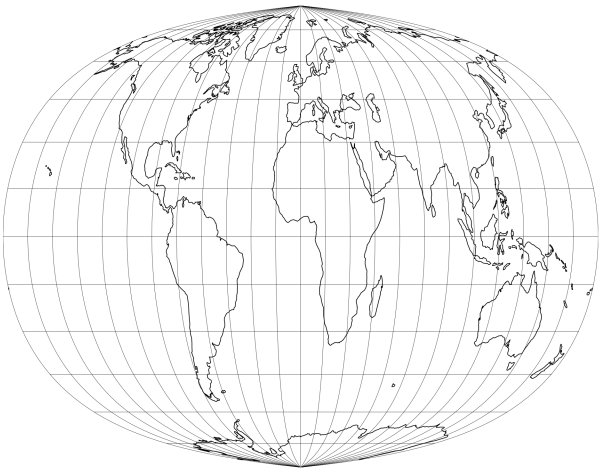


Space oblique for LANDSAT

+proj=lsat +path=2 +lsat=1

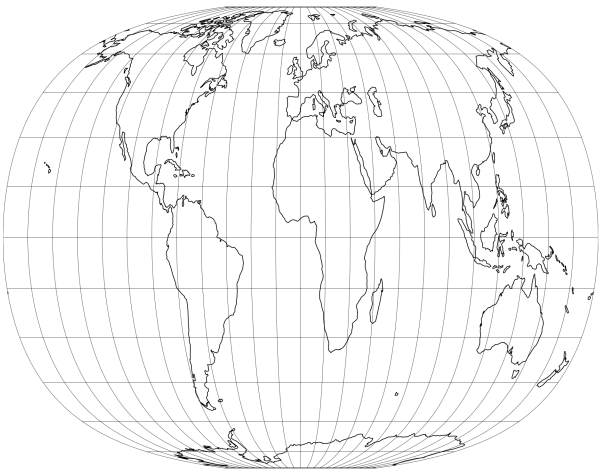
McBryde-Thomas Flat-Polar Sine (No. 1)

+proj=mbt_s



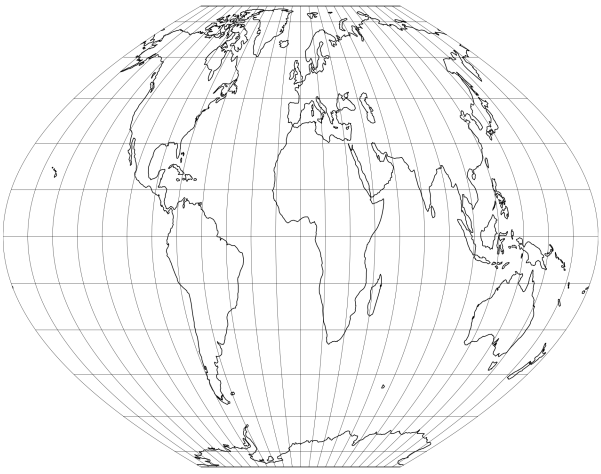
McBryde-Thomas Flat-Pole Sine (No. 2)

+proj=mbt_fps



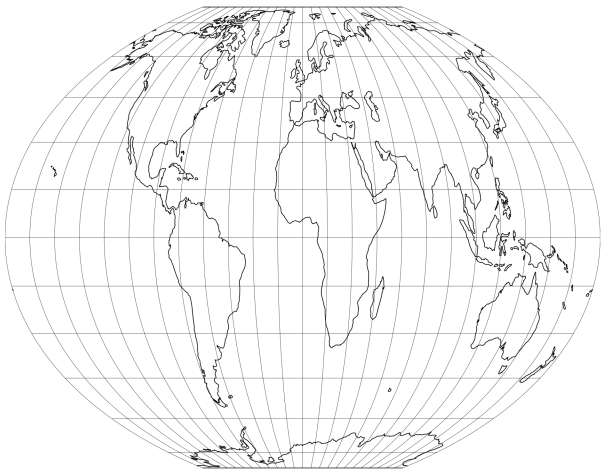
McBride-Thomas Flat-Polar Parabolic

+proj=mbtfpp



McBryde-Thomas Flat-Polar Quartic

+proj=mbtfpq



McBryde-Thomas Flat-Polar Sinusoidal

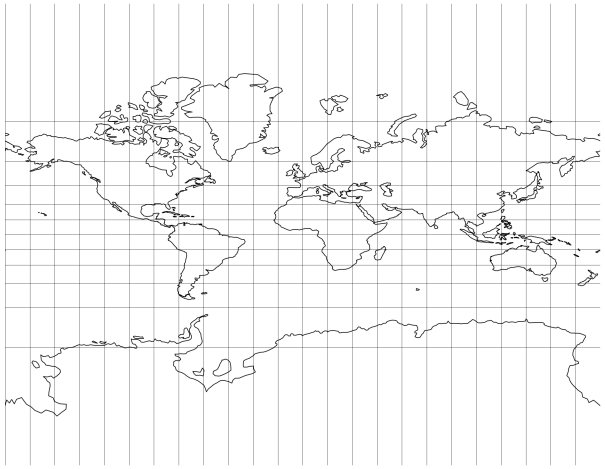
+proj=mbtfps

Mercator

The Mercator projection is a cylindrical map projection that origins from the 15th century. It is widely recognized as the first regularly used map projection. The projection is conformal which makes it suitable for navigational purposes.

Classification	Conformal cylindrical
Available forms	Forward and inverse, spherical and elliptical projection
Defined area	Global, but best used near the equator
Implemented by	Gerald I. Evenden
Options	
+lat_ts	Latitude of true scale. Defaults to 0.0
+k_0	Scaling factor. Defaults to 1.0

+proj=merc



Usage

Applications should be limited to equatorial regions, but is frequently used for navigational charts with latitude of true scale (`+lat_ts`) specified within or near chart's boundaries. Often inappropriately used for world maps since the regions near the poles cannot be shown *[Evenden1995]*.

Example using latitude of true scale:

```
$ echo 56.35 12.32 | proj +proj=merc +lat_ts=56.5
3470306.37    759599.90
```

Example using scaling factor:

```
echo 56.35 12.32 | proj +proj=merc +k_0=2
12545706.61     2746073.80
```

Note that `+lat_ts` and `+k_0` are mutually excusive. If used together, `+lat_ts` takes precedence over `+k_0`.

Mathematical definition

The formulas describing the Mercator projection are all taken from G. Evenden's libproj manuals *[Evenden2005]*.

Spherical form

For the spherical form of the projection we introduce the scaling factor:

$$k_0 = \cos \phi_{ts}$$

Forward projection

$$x = k_0 \lambda$$

$$y = k_0 \ln \left[\tan \left(\frac{\pi}{4} + \frac{\phi}{2} \right) \right]$$

Inverse projection

$$\lambda = \frac{x}{k_0}$$

$$\phi = \frac{\pi}{2} - 2 \arctan \left[e^{-y/k_0} \right]$$

Elliptical form

For the elliptical form of the projection we introduce the scaling factor:

$$k_0 = m \left(\phi_t s \right)$$

where $m \left(\phi \right)$ is the parallel radius at latitude ϕ.

We also use the Isometric Latitude kernel function $t()$.

Note: m() and t() should be described properly on a separate page about the theory of projections on the ellipsoid.

Forward projection

$$x = k_0 \lambda$$

$$y = k_0 \ln t \left(\phi \right)$$

Inverse projection

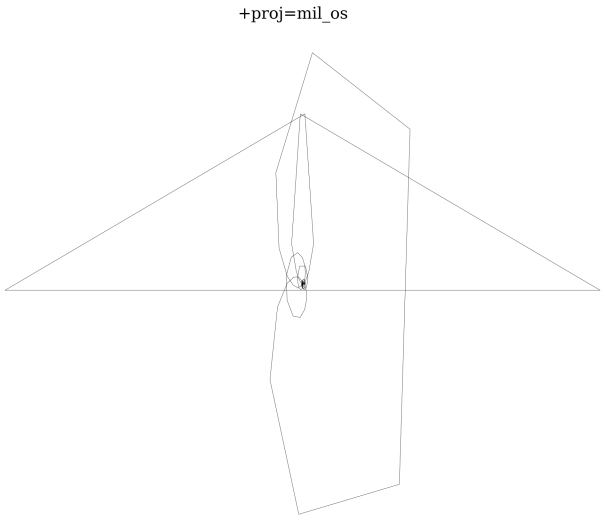
$$\lambda = \frac{x}{k_0}$$

$$\phi = t^{-1} \left[e^{-y/k_0} \right]$$

Further reading

1. Wikipedia
2. Wolfram Mathworld

Miller Oblated Stereographic

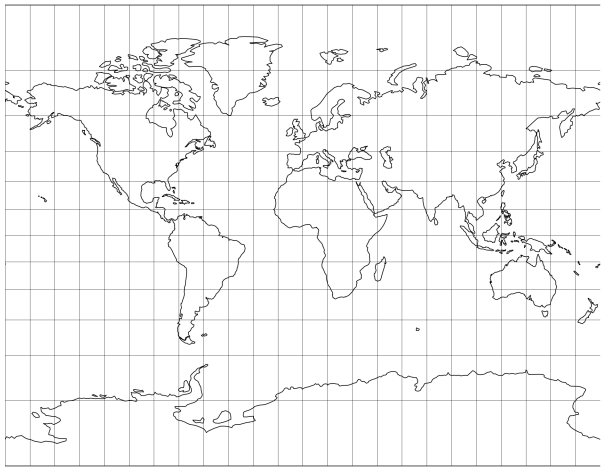


Miller Cylindrical

The Miller cylindrical projection is a modified Mercator projection, proposed by Osborn Maitland Miller in 1942. The latitude is scaled by a factor of $\frac{4}{5}$, projected according to Mercator, and then the result is multiplied by $\frac{5}{4}$ to retain scale along the equator.

Classification	Neither conformal nor equal area cylindrical
Available forms	Forward and inverse spherical
Defined area	Global, but best used near the equator
Implemented by	Gerald I. Evenden
Options	
+lat_0	Latitude of origin (Default to 0)

+proj=mill



Usage

The Miller Cylindrical projection is used for world maps and in several atlases, including the National Atlas of the United States (USGS, 1970, p. 330-331) *[Snyder1987]*.

Example using Central meridian 90°W:

```
$ echo -100 35 | proj +proj=mill +lon_0=90w
-1113194.91      4061217.24
```

Mathematical definition

The formulas describing the Miller projection are all taken from Snyder's manuals *[Snyder1987]*.

Forward projection

$$x = \lambda$$

$$y = 1.25 * \ln\left[\tan\left(\frac{\pi}{4} + 0.4 * \phi\right)\right]$$

Inverse projection

$$\lambda = x$$

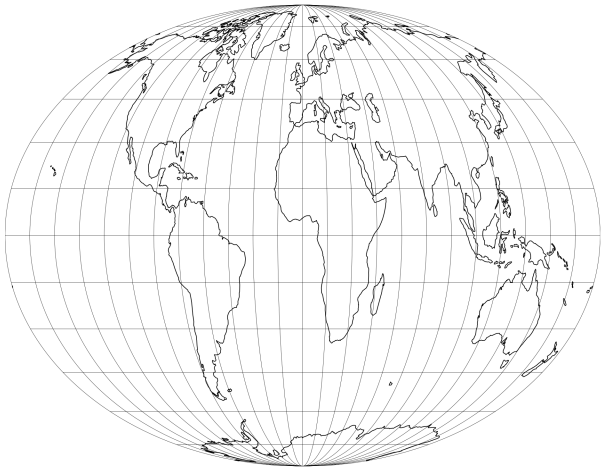
$$\phi = 2.5 * (\arctan\left[e^{0.8*y}\right] - \frac{\pi}{4})$$

Further reading

1. Wikipedia

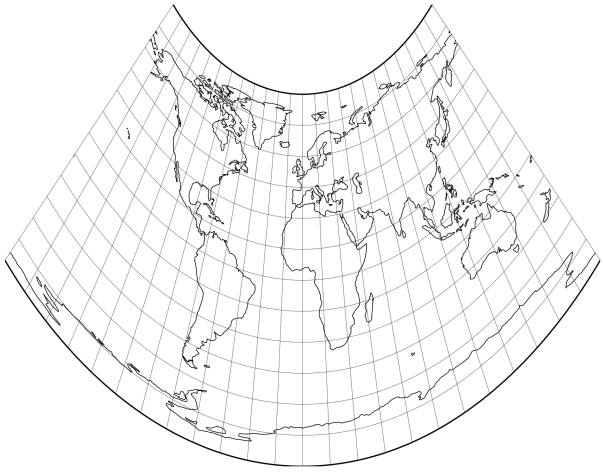
Mollweide

+proj=moll



Murdoch I

+proj=murd1 +lat_1=10 +lat_2=20

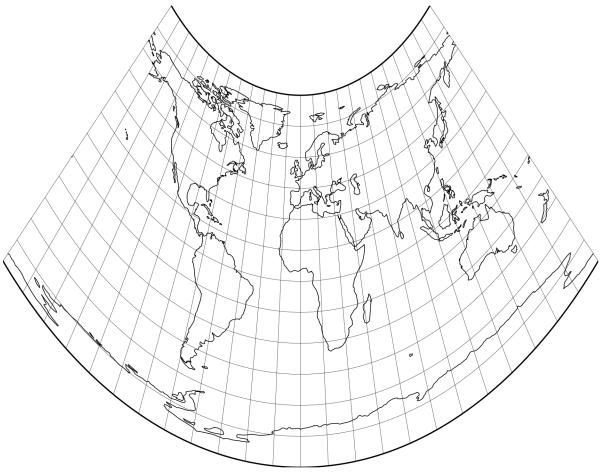


Murdoch II

+proj=murd2 +lat_1=10 +lat_2=20

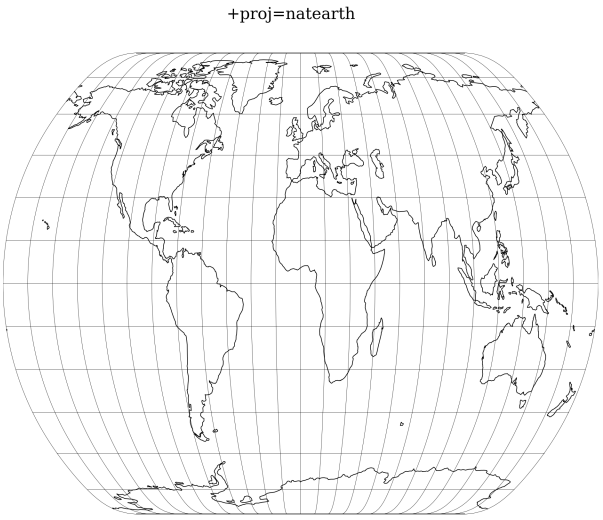
Murdoch III

+proj=murd3 +lat_1=10 +lat_2=20



Natural Earth

Classification	Pseudo cylindrical
Available forms	Forward and inverse, spherical projection
Defined area	Global
Implemented by	Bernhard Jenny
Options	
No special options for this projection	



The Natural Earth projection is intended for making world maps. A distinguishing trait is its slightly rounded corners fashioned to emulate the spherical shape of Earth. The meridians (except for the central meridian) bend acutely inward as they approach the pole lines, giving the projection a hint of three-dimensionality. This bending also suggests that the meridians converge at the poles instead of truncating at the top and bottom edges. The distortion characteristics of the Natural Earth projection compare favorably to other world map projections.

Usage

The Natural Earth projection has no special options so usage is simple. Here is an example of an inverse projection on a sphere with a radius of 7500 m:

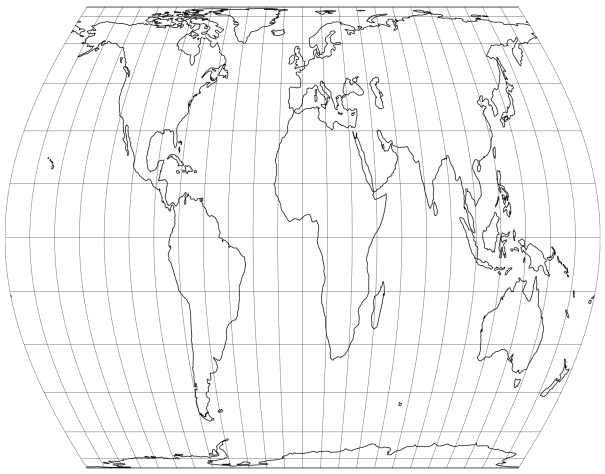
```
$ echo 3500 -8000 | proj -I +proj=natearth +a=7500
37d54'6.091"E  61d23'4.582"S
```

Further reading

1. Wikipedia

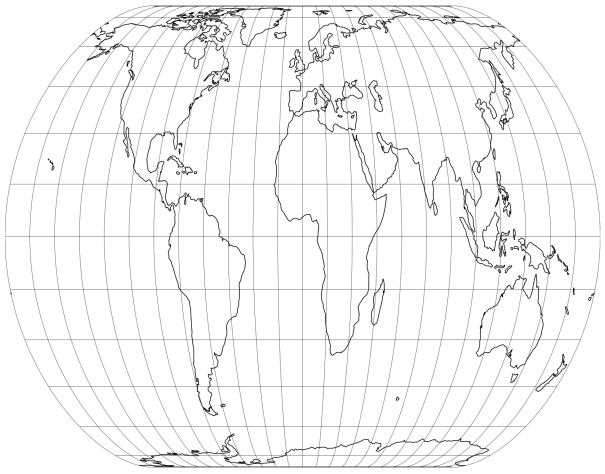
Nell

+proj=nell



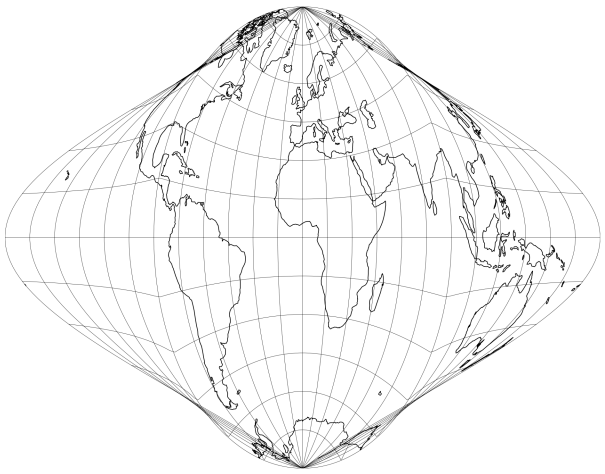
Nell-Hammer

+proj=nell_h



Nicolosi Globular

+proj=nicol

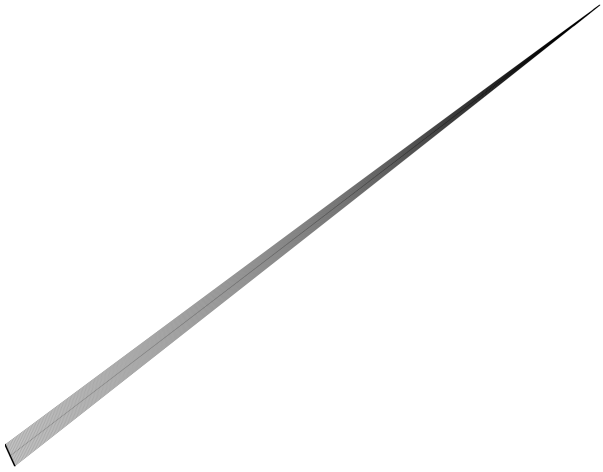


Near-sided perspective

+proj=nsper +h=1

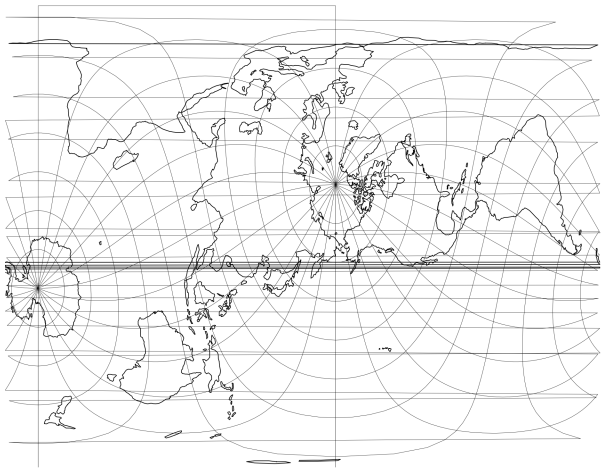
New Zealand Map Grid

+proj=nzmg



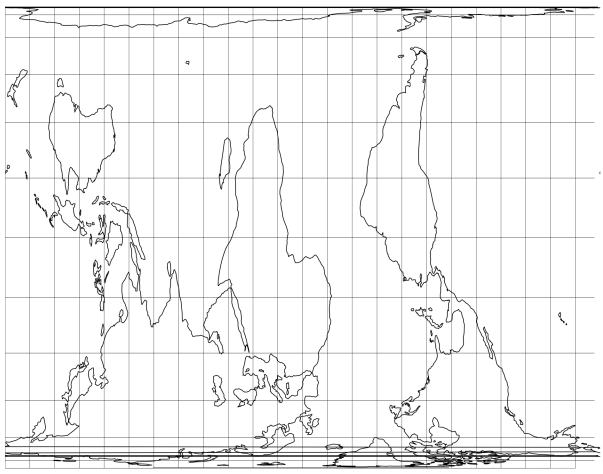
General Oblique Transformation

+proj=ob_tran +o_proj=latlon +o_lon_p=20 +o_lat_p=20 +lon_0=180



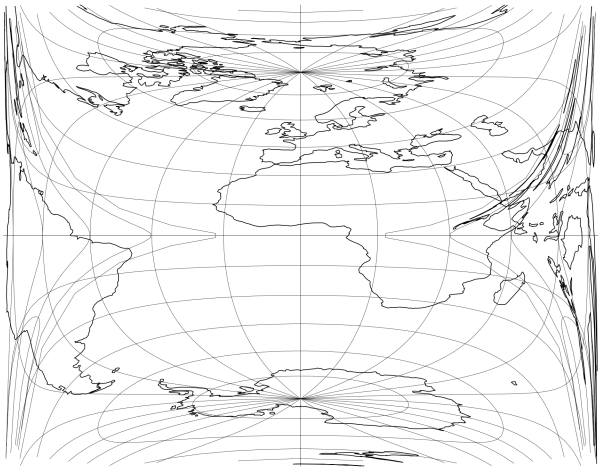
Oblique Cylindrical Equal Area

+proj=ocea



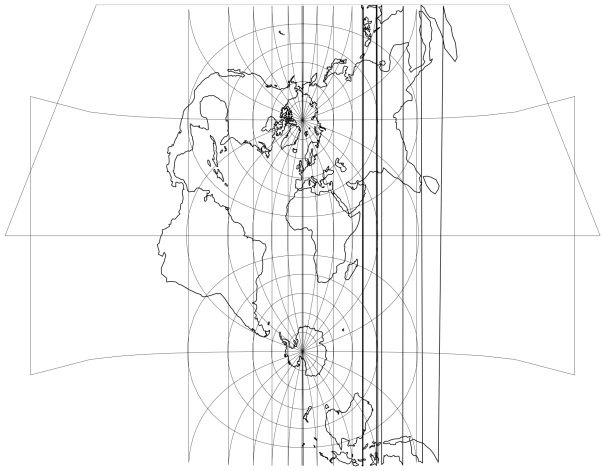
Oblated Equal Area

+proj=oea +m=1 +n=2



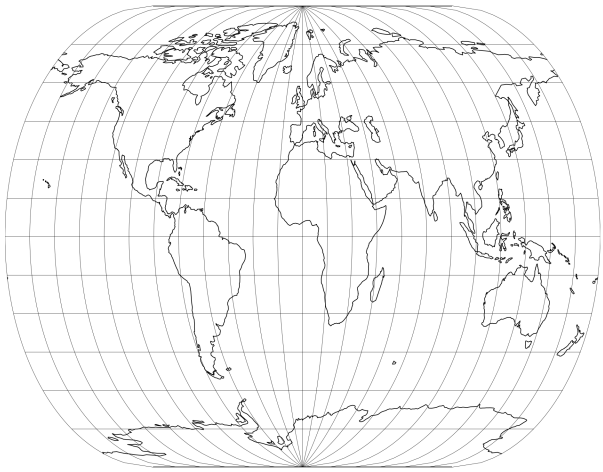
Oblique Mercator

+proj=omerc +lat_1=45 +lat_2=55



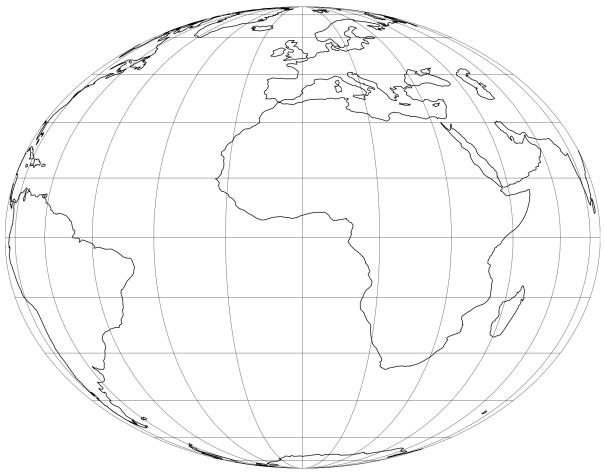
Ortelius Oval

+proj=ortel



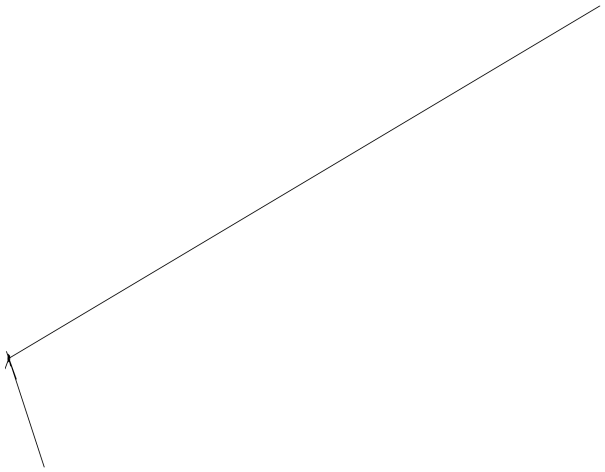
Orthographic

+proj=ortho



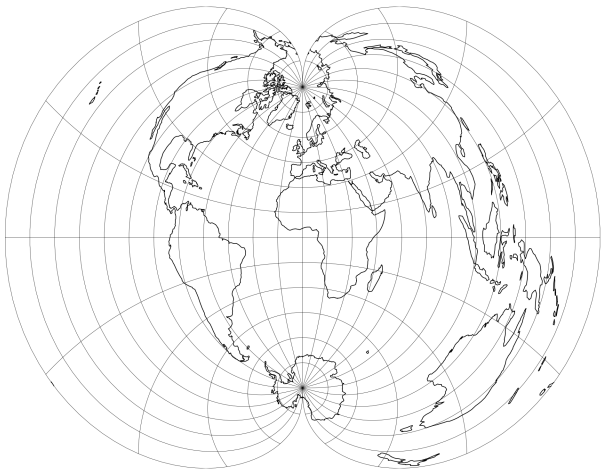
Perspective Conic

+proj=pconic +lat_1=55 +lat_2=75



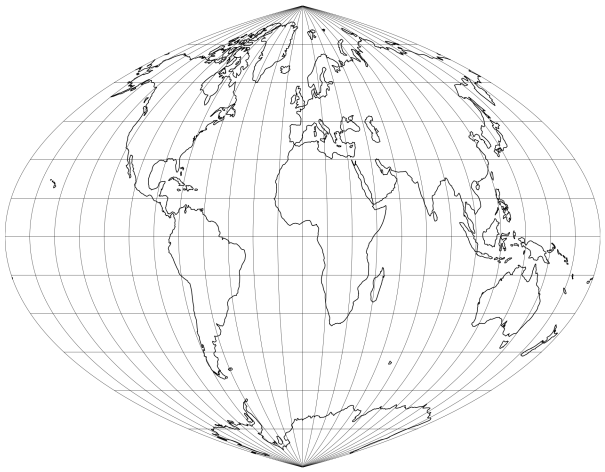
Polyconic (American)

+proj=poly



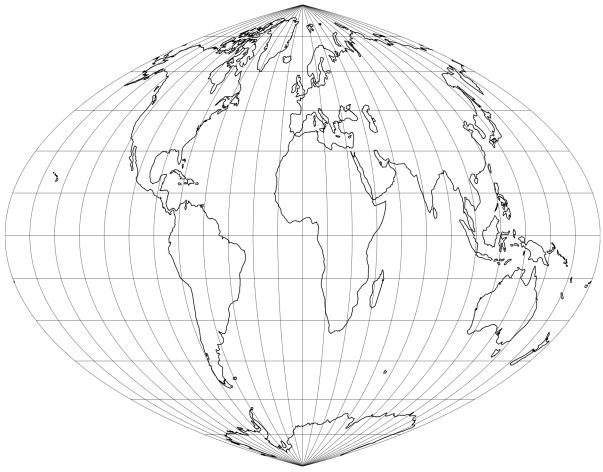
Putnins P1

+proj=putp1



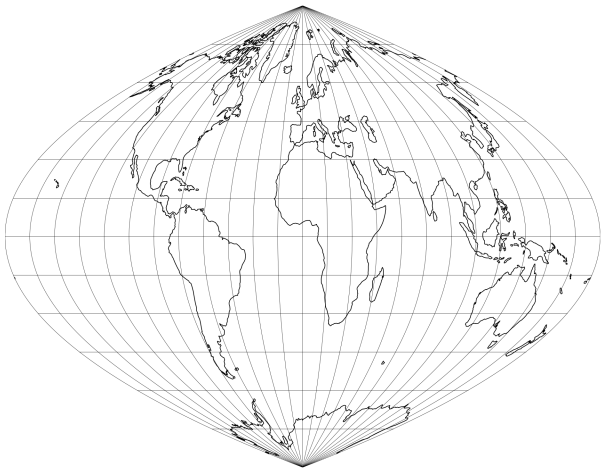
Putnins P2

+proj=putp2



Putnins P3

+proj=putp3

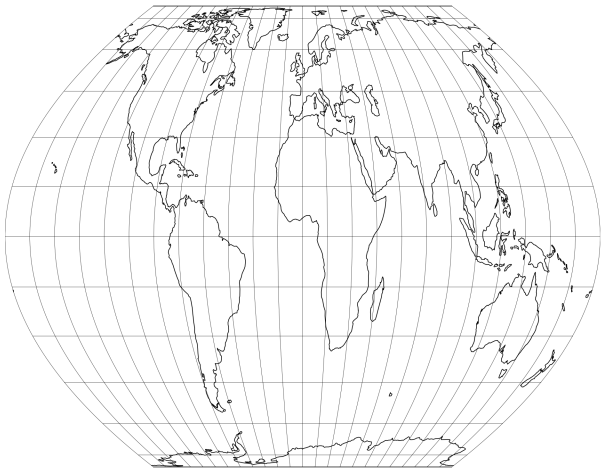


Putnins P3’

+proj=putp3p

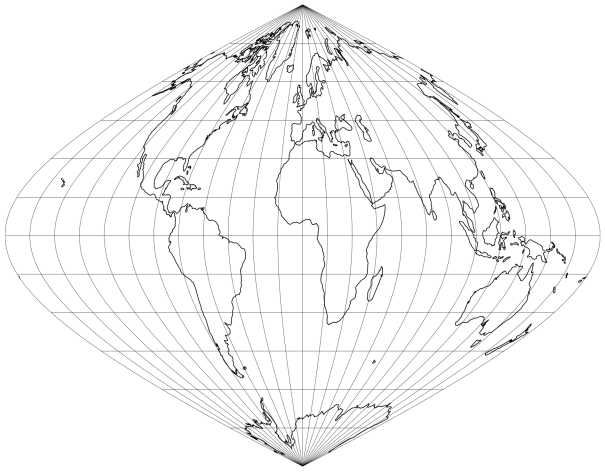
Putnins P4’

+proj=putp4p



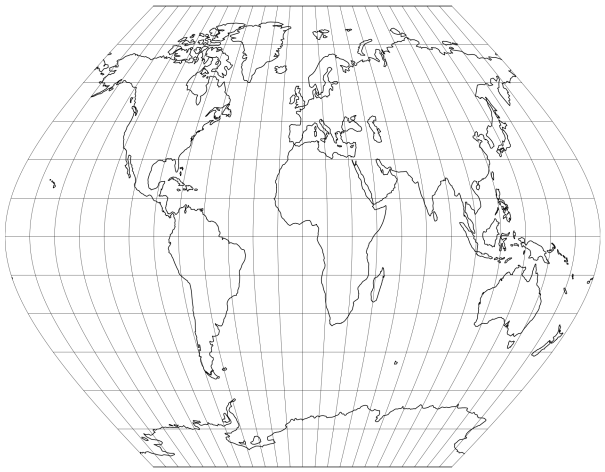
Putnins P5

+proj=putp5



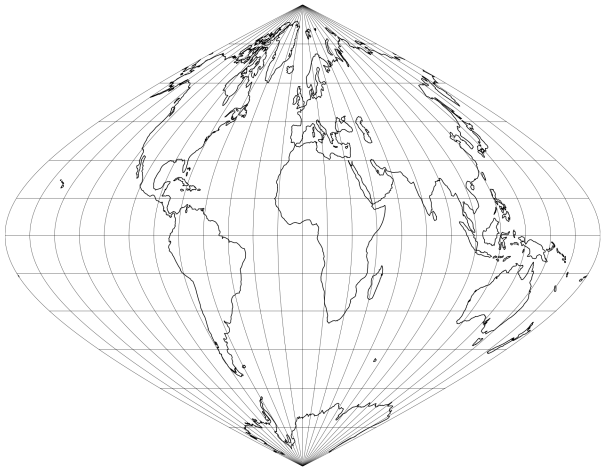
Putnins P5'

+proj=putp5p



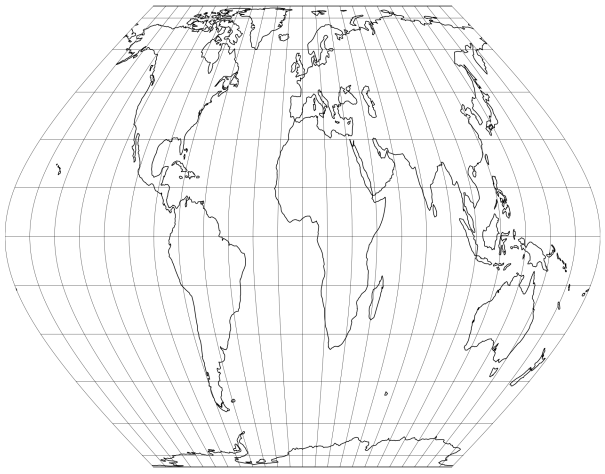
Putnins P6

+proj=putp6

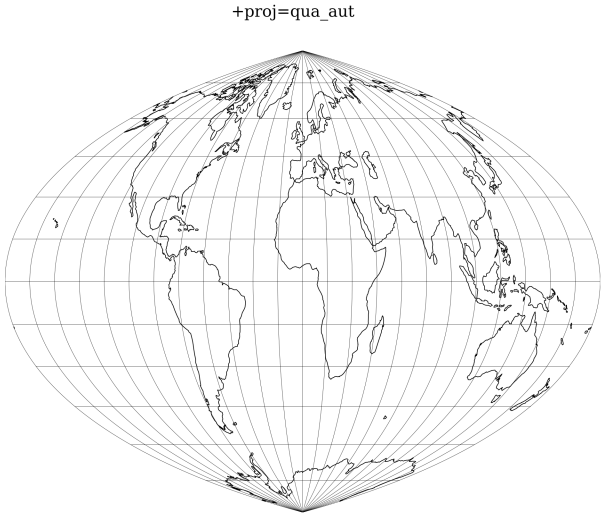


Putnins P6'

+proj=putp6p



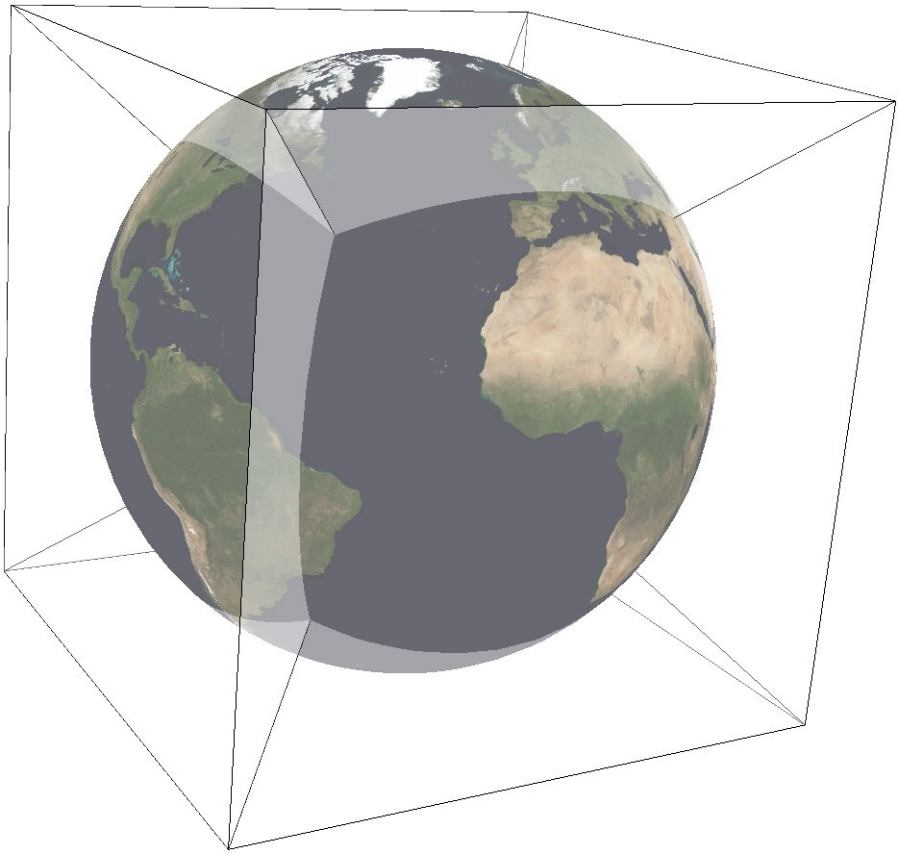
Quartic Authalic



Quadrilateralized Spherical Cube

Classification	Azimuthal
Available forms	Forward and inverse, elliptical projection
Defined area	Global
Implemented by	Martin Lambers
Options	
+lat_0	Latitude (in degrees) of the view position.
+lon_0	Longitude (in degrees) of the view position.

The purpose of the Quadrilateralized Spherical Cube (QSC) projection is to project a sphere surface onto the six sides of a cube:



For this purpose, other alternatives can be used, notably *Gnomonic* or *HEALPix*. However, QSC projection has the following favorable properties:

It is an equal-area projection, and at the same time introduces only limited angular distortions. It treats all cube sides equally, i.e. it does not use different projections for polar areas and equatorial areas. These properties make QSC projection a good choice for planetary-scale terrain rendering. Map data can be organized in quadtree structures for each cube side. See *[LambersKolb2012]* for an example.

The QSC projection was introduced by *[ONeilLaubscher1976]*, building on previous work by *[ChanONeil1975]*. For clarity: The earlier QSC variant described in *[ChanONeil1975]* became known as the COBE QSC since it was used by the NASA Cosmic Background Explorer (COBE) project; it is an approximately equal-area projection and is not the same as the QSC projection.

See also *[CalabrettaGreisen2002]* Sec. 5.6.2 and 5.6.3 for a description of both and some analysis.

In this implementation, the QSC projection projects onto one side of a circumscribed cube. The cube side is selected by choosing one of the following six projection centers:

`+lat_0=0 +lon_0=0`	front cube side
`+lat_0=0 +lon_0=90`	right cube side
`+lat_0=0 +lon_0=180`	back cube side
`+lat_0=0 +lon_0=-90`	left cube side
`+lat_0=90`	top cube side
`+lat_0=-90`	bottom cube side

Furthermore, this implementation allows the projection to be applied to ellipsoids. A preceding shift to a sphere is performed automatically; see *[LambersKolb2012]* for details.

Usage

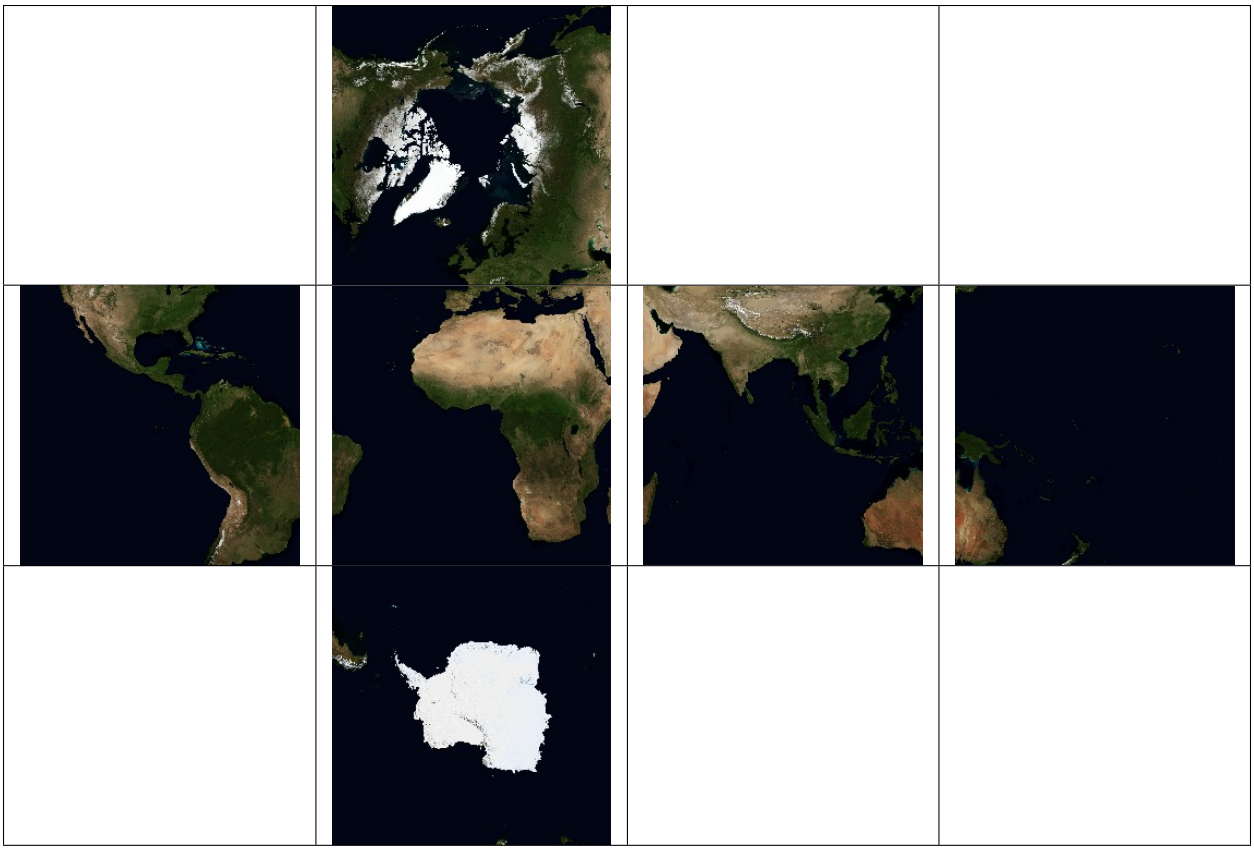
The following example uses QSC projection via GDAL to create the six cube side maps from a world map for the WGS84 ellipsoid:

```
gdalwarp -t_srs "+wktext +proj=qsc +units=m +ellps=WGS84  +lat_0=0 +lon_0=0"        \
    -wo SOURCE_EXTRA=100 -wo SAMPLE_GRID=YES -te -6378137 -6378137 6378137 6378137  \
    worldmap.tiff frontside.tiff

gdalwarp -t_srs "+wktext +proj=qsc +units=m +ellps=WGS84 +lat_0=0 +lon_0=90"        \
    -wo SOURCE_EXTRA=100 -wo SAMPLE_GRID=YES -te -6378137 -6378137 6378137 6378137  \
    worldmap.tiff rightside.tiff

gdalwarp -t_srs "+wktext +proj=qsc +units=m +ellps=WGS84 +lat_0=0 +lon_0=180"       \
    -wo SOURCE_EXTRA=100 -wo SAMPLE_GRID=YES -te -6378137 -6378137 6378137 6378137  \
    worldmap.tiff backside.tiff

gdalwarp -t_srs "+wktext +proj=qsc +units=m +ellps=WGS84 +lat_0=0 +lon_0=-90"       \
    -wo SOURCE_EXTRA=100 -wo SAMPLE_GRID=YES -te -6378137 -6378137 6378137 6378137  \
    worldmap.tiff leftside.tiff

gdalwarp -t_srs "+wktext +proj=qsc +units=m +ellps=WGS84 +lat_0=90 +lon_0=0"        \
    -wo SOURCE_EXTRA=100 -wo SAMPLE_GRID=YES -te -6378137 -6378137 6378137 6378137  \
    worldmap.tiff topside.tiff

gdalwarp -t_srs "+wktext +proj=qsc +units=m +ellps=WGS84 +lat_0=-90 +lon_0=0"       \
    -wo SOURCE_EXTRA=100 -wo SAMPLE_GRID=YES -te -6378137 -6378137 6378137 6378137  \
    worldmap.tiff bottomside.tiff
```

Explanation:

- QSC projection is selected with `+wktext +proj=qsc`.
- The WGS84 ellipsoid is specified with `+ellps=WGS84`.
- The cube side is selected with `+lat_0=... +lon_0=...`.
- The `-wo` options are necessary for GDAL to avoid holes in the output maps.
- The `-te` option limits the extends of the output map to the major axis diameter (from -radius to +radius in both x and y direction). These are the dimensions of one side of the circumscribing cube.

The resulting images can be laid out in a grid like below.

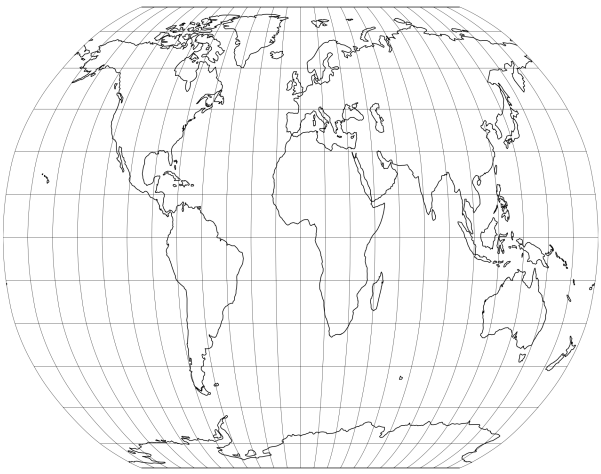


Further reading

1. Wikipedia
2. NASA

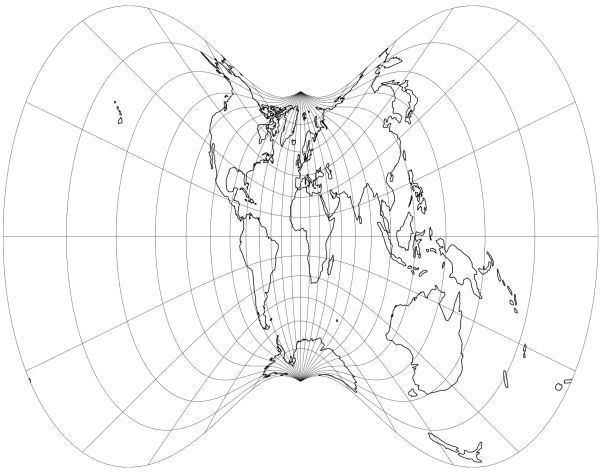
Robinson

+proj=robin



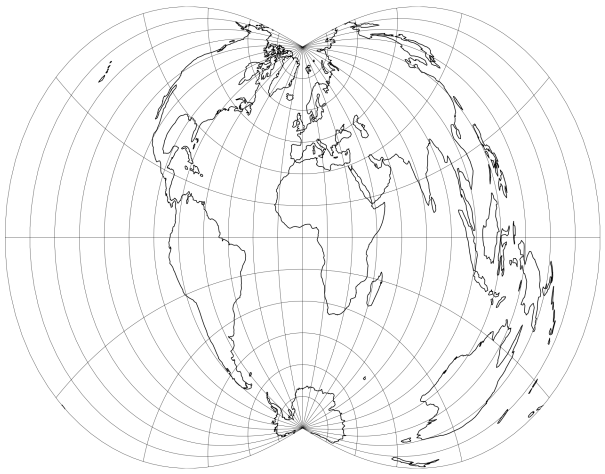
Roussilhe Stereographic

+proj=rouss



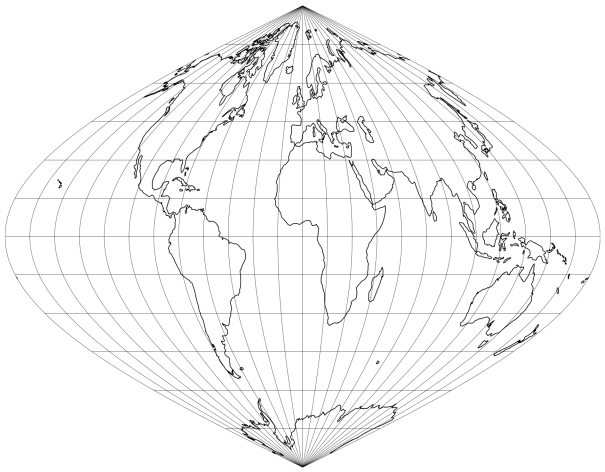
Rectangular Polyconic

+proj=rpoly



Sinusoidal (Sanson-Flamsteed)

+proj=sinu



MacBryde and Thomas developed generalized formulas for sever of the pseudocylindricals with sinusoidal meridians:

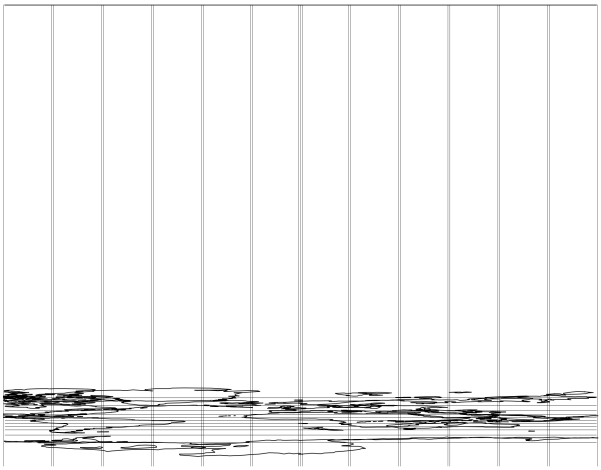
$$x = C\lambda(m + cos\theta)/(m + 1)$$

$$y = C\theta$$

$$C = \sqrt{(m + 1)/n}$$

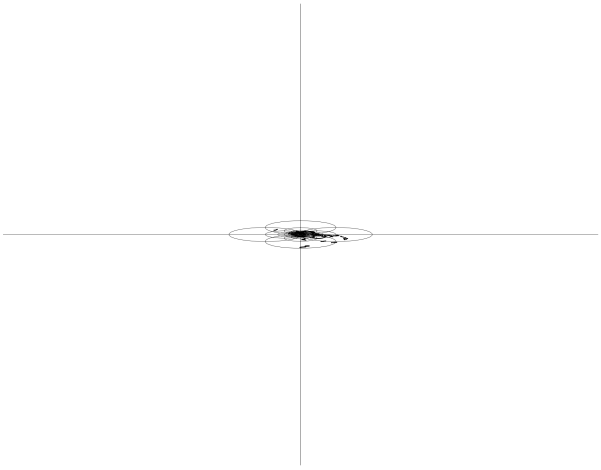
Swiss. Obl. Mercator

+proj=somerc



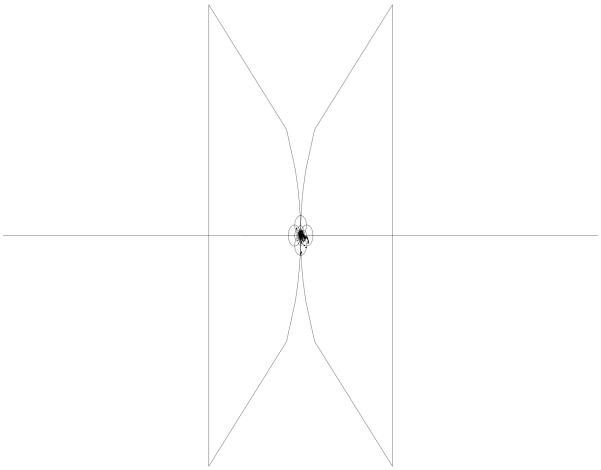
Stereographic

+proj=stere



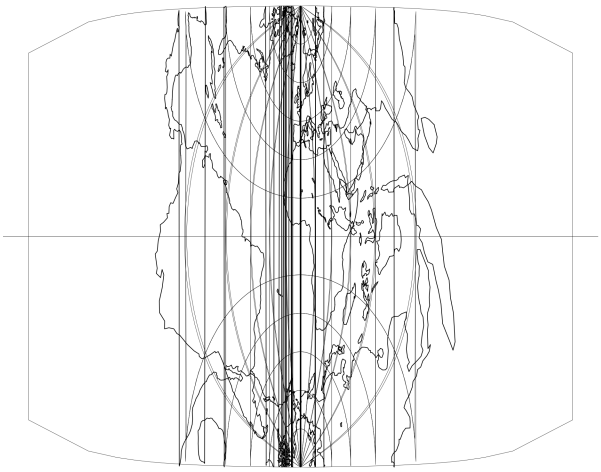
Oblique Stereographic Alternative

+proj=sterea



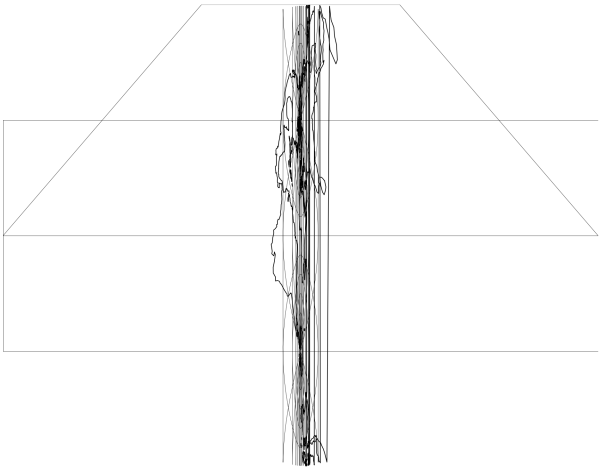
Gauss-Schreiber Transverse Mercator (aka Gauss-Laborde Reunion)

+proj=gstmerc



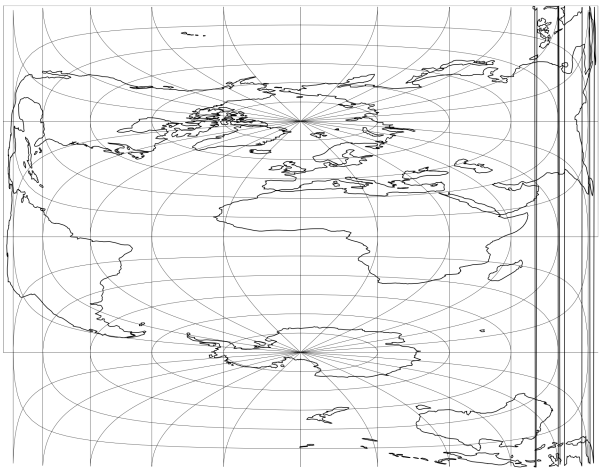
Transverse Central Cylindrical

+proj=tcc



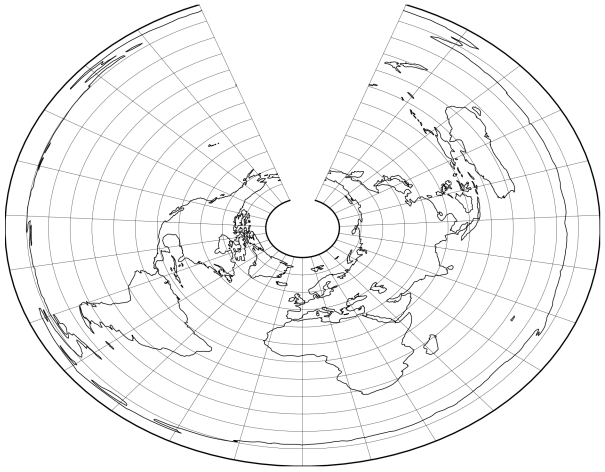
Transverse Cylindrical Equal Area

+proj=tcea



Tissot

+proj=tissot +lat_1=60 +lat_2=65

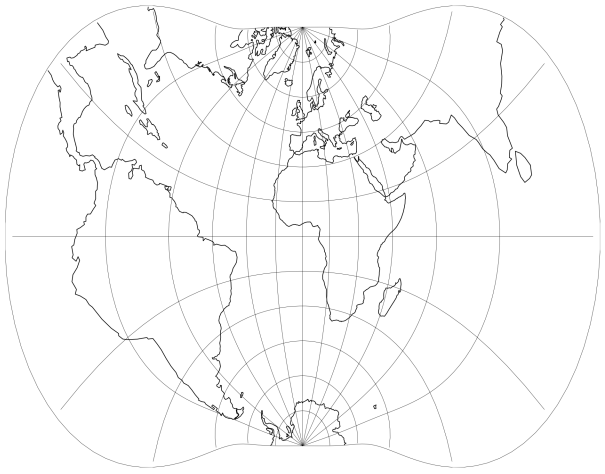


Transverse Mercator

The transverse Mercator projection in its various forms is the most widely used projected coordinate system for world topographical and offshore mapping.

Classification	Transverse and oblique cylindrical
Available forms	Forward and inverse, Spherical and Elliptical
Defined area	Global, but reasonably accurate only within 15 degrees of the central meridian
Implemented by	Gerald I. Evenden
Options	
+lat_0	Latitude of origin (Default to 0)
+k0	Scale factor at natural origin (Default to 1)

+proj=tmerc



Usage

Prior to the development of the Universal Transverse Mercator coordinate system, several European nations demonstrated the utility of grid-based conformal maps by mapping their territory during the interwar period. Calculating the distance between two points on these maps could be performed more easily in the field (using the Pythagorean theorem) than was possible using the trigonometric formulas required under the graticule-based system of latitude and longitude. In the post-war years, these concepts were extended into the Universal Transverse Mercator/Universal Polar Stereographic (UTM/UPS) coordinate system, which is a global (or universal) system of grid-based maps.

The following table gives special cases of the Transverse Mercator projection.

Projection Name	Areas	Central meridian	Zone width	Scale Factor
Transverse Mercator	World wide	Various	less than 6°	Various
Transverse Mercator south oriented	Southern Africa	2° intervals E of 11°E	2°	1.000
UTM North hemisphere	World wide equator to 84°N	6° intervals E & W of 3° E & W	Always 6°	0.9996
UTM South hemisphere	World wide north of 80°S to equator	6° intervals E & W of 3° E & W	Always 6°	0.9996
Gauss-Kruger	Former USSR, Yugoslavia, Germany, S. America, China	Various, according to area	Usually less than 6°, often less than 4°	1.0000
Gauss Boaga	Italy	Various, according to area	6°	0.9996

Example using Gauss-Kruger on Germany area (aka EPSG:31467)

```
$ echo 9 51 | proj +proj=tmerc +lat_0=0 +lon_0=9 +k=1 +x_0=3500000 +y_0=0␣
↪+ellps=bessel +datum=potsdam +units=m +no_defs
3500000.00  5651505.56
```

Example using Gauss Boaga on Italy area (EPSG:3004)

```
$ echo 15 42 | proj +proj=tmerc +lat_0=0 +lon_0=15 +k=0.9996 +x_0=2520000 +y_0=0␣
↪+ellps=intl +units=m +no_defs
2520000.00  4649858.60
```

Mathematical definition

The formulas describing the Transverse Mercator are all taken from Evenden's *[Evenden2005]*.

ϕ_0 is the latitude of origin that match the center of the map. It can be set with `+lat_0`.

k_0 is the scale factor at the natural origin (on the central meridian). It can be set with `+k_0`.

$M(\phi)$ is the meridianal distance.

Spherical form

Forward projection

$$B = \cos\phi \sin\lambda$$

$$x = \frac{k_0}{2} \ln(\frac{1+B}{1-B})$$

$$y = k_0(\arctan(\frac{\tan(\phi)}{\cos\lambda}) - \phi_0)$$

Inverse projection

$$D = \frac{y}{k_0} + \phi_0$$

$$x' = \frac{x}{k_0}$$

$$\phi = \arcsin(\frac{\sin D}{\cosh x'})$$

$$\lambda = \arctan(\frac{\sinh x'}{\cos D})$$

Elliptical form

Forward projection

$$N = \frac{k_0}{(1 - e^2 \sin^2\phi)^{1/2}}$$

$$R = \frac{k_0(1-e^2)}{(1 - e^2 \sin^2\phi)^{3/2}}$$

$$t = \tan(\phi)$$

$$\eta = \frac{e^2}{1-e^2} cos^2\phi$$

$$\begin{aligned} x = {} & k_0\lambda\cos\phi \\ & + \frac{k_0\lambda^3\cos^3\phi}{3!}(1 - t^2 + \eta^2) \\ & + \frac{k_0\lambda^5\cos^5\phi}{5!}(5 - 18t^2 + t^4 + 14\eta^2 - 58t^2\eta^2) \\ & + \frac{k_0\lambda^7\cos^7\phi}{7!}(61 - 479t^2 + 179t^4 - t^6) \end{aligned}$$

$$\begin{aligned} y = {} & M(\phi) \\ & + \frac{k_0\lambda^2\sin(\phi)\cos\phi}{2!} \\ & + \frac{k_0\lambda^4\sin(\phi)\cos^3\phi}{4!}(5 - t^2 + 9\eta^2 + 4\eta^4) \\ & + \frac{k_0\lambda^6\sin(\phi)\cos^5\phi}{6!}(61 - 58t^2 + t^4 + 270\eta^2 - 330t^2\eta^2) \\ & + \frac{k_0\lambda^8\sin(\phi)\cos^7\phi}{8!}(1385 - 3111t^2 + 543t^4 - t^6) \end{aligned}$$

Inverse projection

$$\phi_1 = M^-1(y)$$

$$N_1 = \frac{k_0}{1 - e^2 \sin^2 \phi_1)^{1/2}}$$

$$R_1 = \frac{k_0(1 - e^2)}{(1 - e^2 \sin^2 \phi_1)^{3/2}}$$

$$t_1 = \tan(\phi_1)$$

$$\eta_1 = \frac{e^2}{1 - e^2} cos^2 \phi_1$$

$$\begin{aligned}
\phi = \phi_1 & \\
& - \frac{t_1 x^2}{2! R_1 N_1} \\
& + \frac{t_1 x^4}{4! R_1 N_1^3}(5 + 3t_1^2 + \eta_1^2 - 4\eta_1^4 - 9\eta_1^2 t_1^2) \\
& - \frac{t_1 x^6}{6! R_1 N_1^5}(61 + 90t_1^2 + 46\eta_1^2 + 45t_1^4 - 252t_1^2\eta_1^2) \\
& + \frac{t_1 x^8}{8! R_1 N_1^7}(1385 + 3633t_1^2 + 4095t_1^4 + 1575t_1^6)
\end{aligned}$$

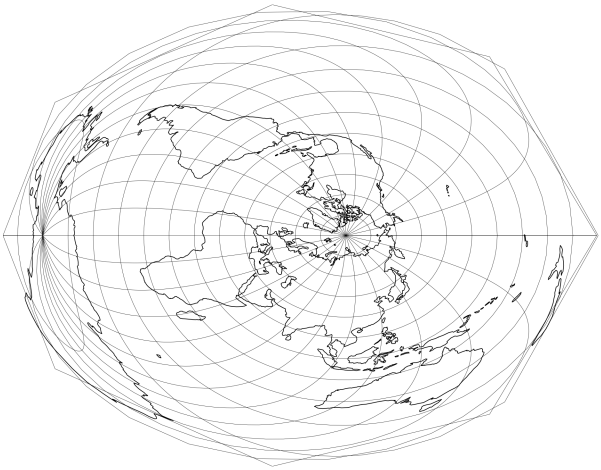
$$\begin{aligned}
\lambda = & \frac{x}{\cos \phi N_1} \\
& - \frac{x^3}{3! \cos \phi N_1^3}(1 + 2t_1^2 + \eta_1^2) \\
& + \frac{x^5}{5! \cos \phi N_1^5}(5 + 6\eta_1^2 + 28t_1^2 - 3\eta_1^2 + 8t_1^2\eta_1^2) \\
& - \frac{x^7}{7! \cos \phi N_1^7}(61 + 662t_1^2 + 1320t_1^4 + 720t_1^6)
\end{aligned}$$

Further reading

1. Wikipedia
2. EPSG, POSC literature pertaining to Coordinate Conversions and Transformations including Formulas

Two Point Equidistant

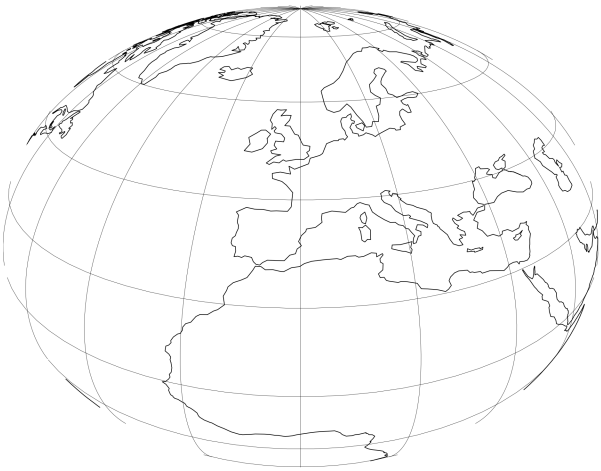
+proj=tpeqd +lat_1=60 +lat_2=65



Tilted perspective

Classification	Azimuthal
Available forms	Forward and inverse, spherical projection
Defined area	Global
Implemented by	Gerald I. Evenden
Options	
+h	Height (in meters) above the surface. Required.
+azi	Bearing (in degrees) from due north.
+tilt	Angle (in degrees) away from nadir.
+lat_0	Latitude (in degrees) of the view position.
+lon_0	Longitude (in degrees) of the view position.

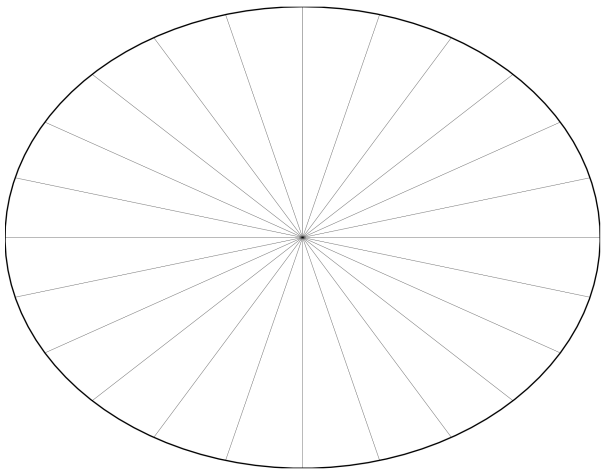
+proj=tpers +h=5500000 +lat_0=40



Tilted Perspective is similar to *Near-sided perspective* (`nsper`) in that it simulates a perspective view from a hight. Where `nsper` projects onto a plane tangent to the surface, Tilted Perspective orients the plane towards the direction of the view. Thus, extra parameters azi and tilt are required beyond *nsper*‘‘s `h`. As with `nsper`, `lat_0` & `lon_0` are also required for satellite position.

Universal Polar Stereographic

+proj=ups

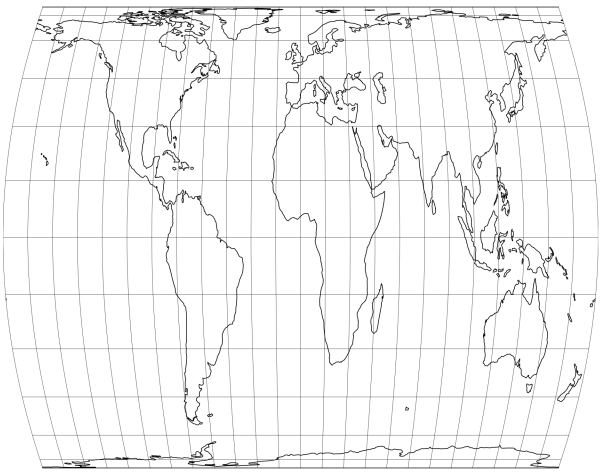


Urmaev V

+proj=urm5

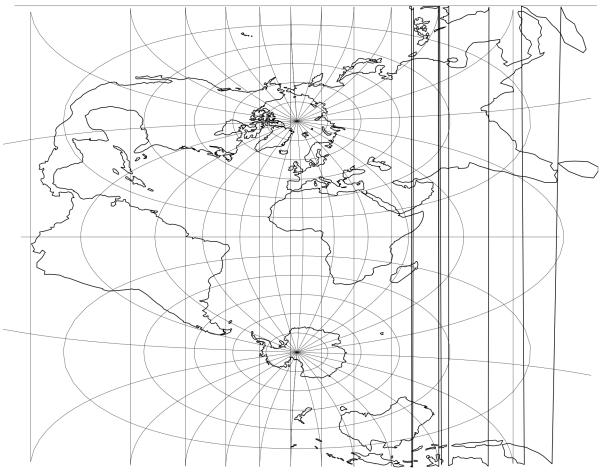
Urmaev Flat-Polar Sinusoidal

+proj=urmfps +n=0.5



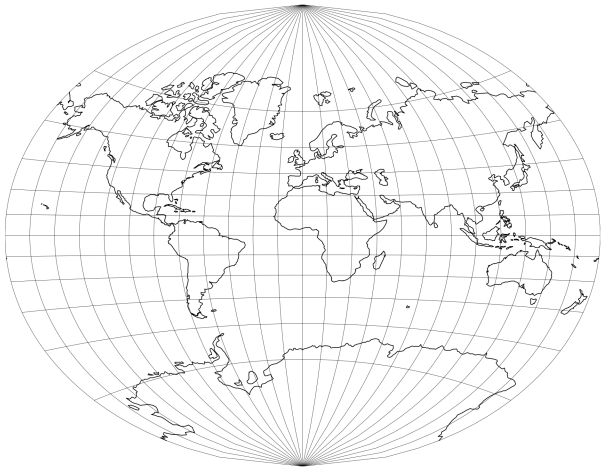
Universal Transverse Mercator (UTM)

+proj=utm



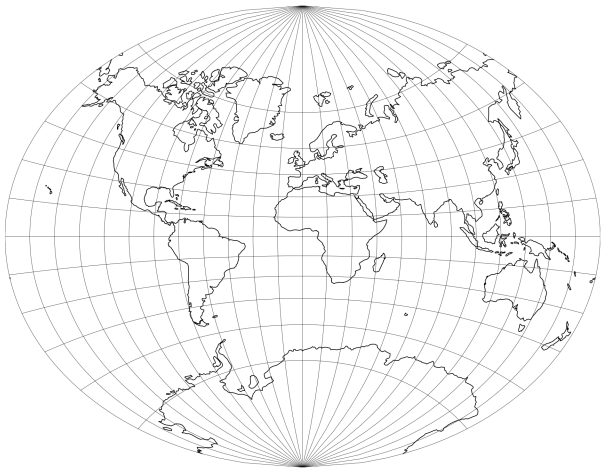
van der Grinten (I)

+proj=vandg



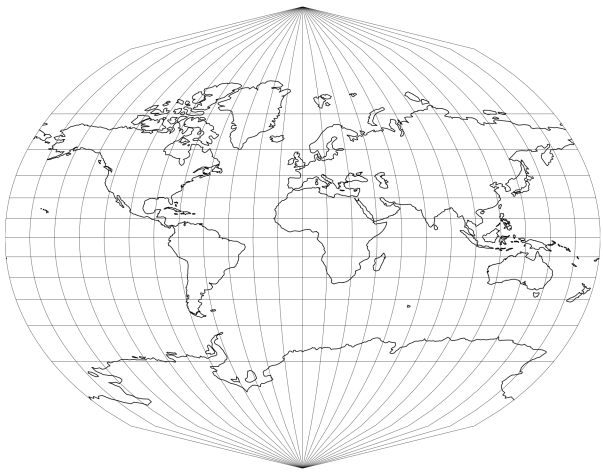
van der Grinten II

+proj=vandg2



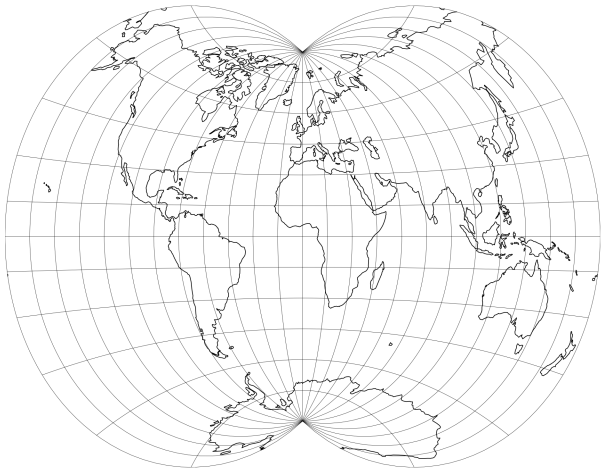
van der Grinten III

+proj=vandg3



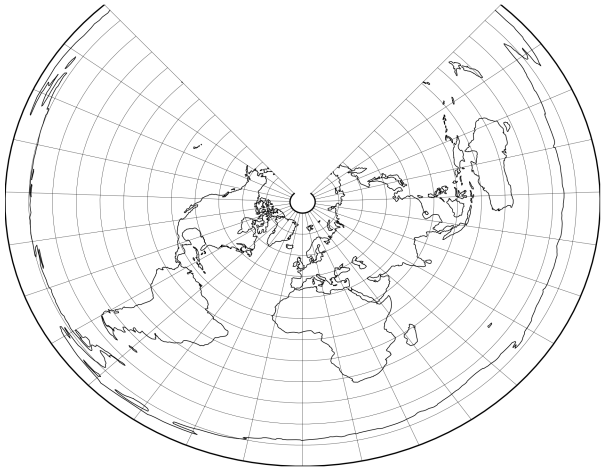
van der Grinten IV

+proj=vandg4



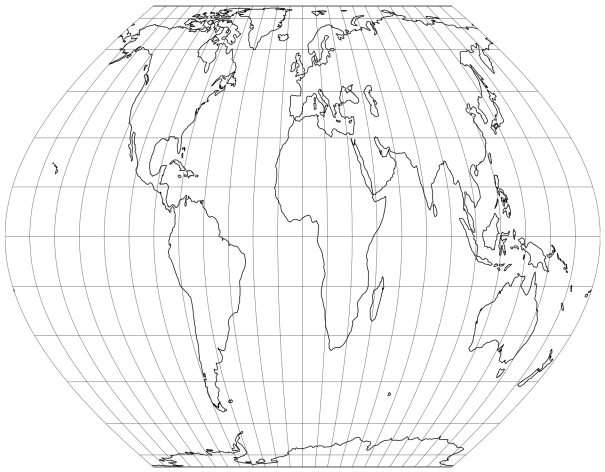
Vitkovsky I

+proj=vitk1 +lat_1=45 +lat_2=55

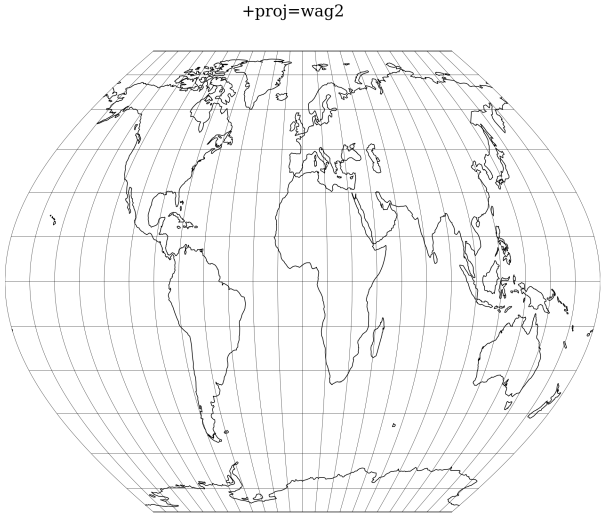


Wagner I (Kavraisky VI)

+proj=wag1



Wagner II



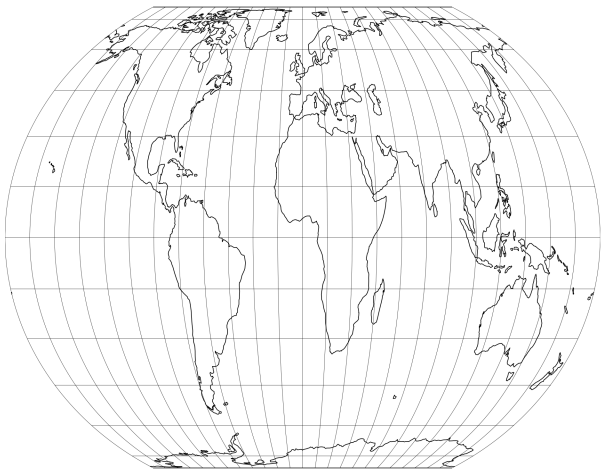
$$x = 0.92483\lambda\cos\theta$$
$$y = 1.38725\theta$$
$$\sin\theta = 0.88022\sin(0.8855\phi)$$

Wagner III

$$x = [\cos\phi_{ts}/\cos(2\phi_{ts}/3)]\lambda\cos(2\phi/3)$$
$$y = \phi$$

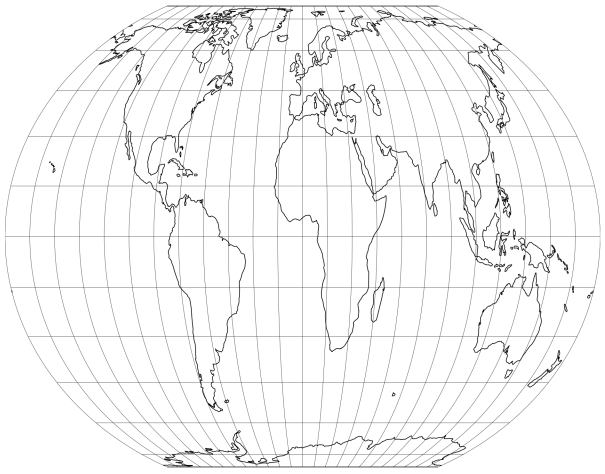
Wagner IV

+proj=wag4



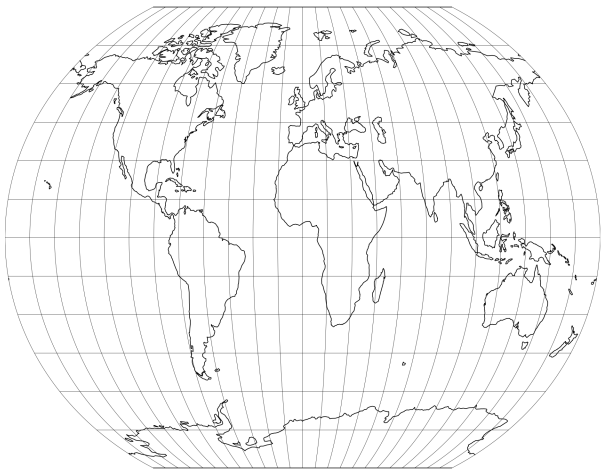
Wagner V

+proj=wag5



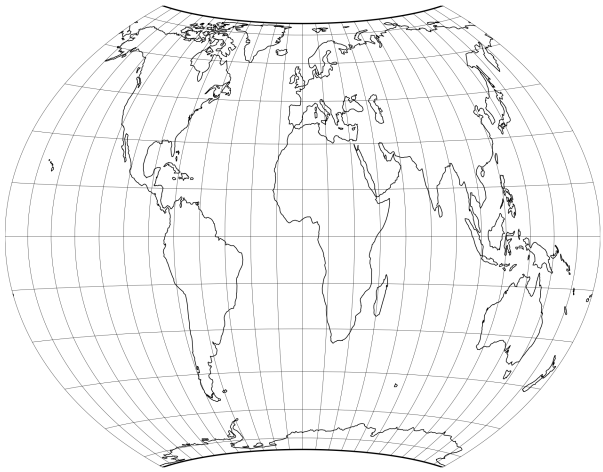
Wagner VI

+proj=wag6



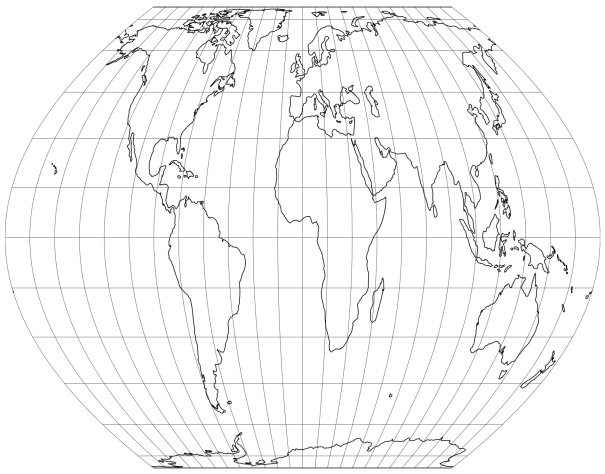
Wagner VII

+proj=wag7



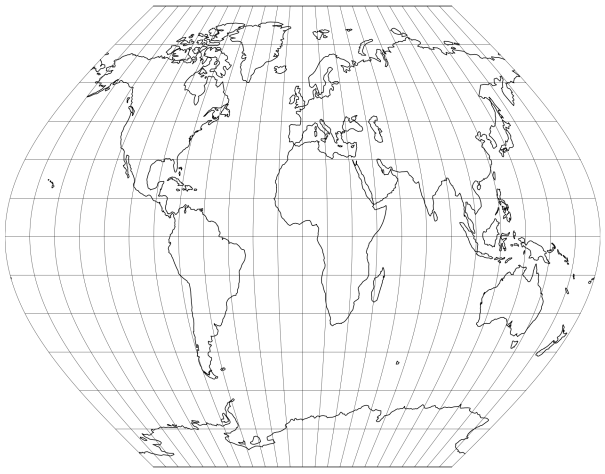
Werenskiold I

+proj=weren



Winkel I

+proj=wink1



Winkel II

+proj=wink2

Winkel Tripel

+proj=wintri

Parameters

Date 01/28/2016

Contents

This document attempts to describe a variety of the PROJ.4 parameters which can be applied to all, or many coordinate system definitions. This document does not attempt to describe the parameters particular to particular projection types. Some of these can be found in the GeoTIFF Projections Transform List. The definitive documentation for most parameters is Gerald's original documentation available from the main PROJ.4 page.

Parameter list

Common parameters:

(this PROJ.4 distribution including *cs2cs* and datum support)

```
+a          Semimajor radius of the ellipsoid axis
+alpha      ? Used with Oblique Mercator and possibly a few others
+axis       Axis orientation (new in 4.8.0)
+b          Semiminor radius of the ellipsoid axis
+datum      Datum name (see `proj -ld`)
+ellps      Ellipsoid name (see `proj -le`)
+k          Scaling factor (old name)
+k_0        Scaling factor (new name)
+lat_0      Latitude of origin
+lat_1      Latitude of first standard parallel
+lat_2      Latitude of second standard parallel
+lat_ts     Latitude of true scale
+lon_0      Central meridian
+lonc       ? Longitude used with Oblique Mercator and possibly a few others
+lon_wrap   Center longitude to use for wrapping (see below)
+nadgrids   Filename of NTv2 grid file to use for datum transforms (see below)
+no_defs    Don't use the /usr/share/proj/proj_def.dat defaults file
+over       Allow longitude output outside -180 to 180 range, disables wrapping (see␣
↪below)
+pm         Alternate prime meridian (typically a city name, see below)
+proj       Projection name (see `proj -l`)
+south      Denotes southern hemisphere UTM zone
+to_meter   Multiplier to convert map units to 1.0m
+towgs84    3 or 7 term datum transform parameters (see below)
```

```
+units     meters, US survey feet, etc.
+vto_meter vertical conversion to meters.
+vunits    vertical units.
+x_0       False easting
+y_0       False northing
+zone      UTM zone
```

Extended list provided by Gerald Evenden "grepped out of the RCS directory".

(libproj4 by G.E.; no datum support)

```
+a         Semimajor radius of the ellipsoid axis
+alpha     ? Used with Oblique Mercator and possibly a few others
+azi
+b         Semiminor radius of the ellipsoid axis
+belgium
+beta
+czech
+e         Eccentricity of the ellipsoid = sqrt(1 - b^2/a^2) = sqrt( f*(2-f) )
+ellps     Ellipsoid name (see `proj -le`)
+es        Eccentricity of the ellipsoid squared
+f         Flattening of the ellipsoid = 1-sqrt(1-e^2) (often presented as an inverse,
↪ e.g. 1/298)
+geoc
+guam
+h
+k         Scaling factor (old name)
+K
+k_0       Scaling factor (new name)
+lat_0     Latitude of origin
+lat_1     Latitude of first standard parallel
+lat_2     Latitude of second standard parallel
+lat_b
+lat_t
+lat_ts    Latitude of true scale
+lon_0     Central meridian
+lon_1
+lon_2
+lonc      ? Longitude used with Oblique Mercator and possibly a few others
+lsat
+m
+M
+n
+no_cut
+no_off    No offset. If present, do not offset origin to center of projection. Only␣
↪used in Oblique Mercator projection.
+no_uoff   Backwards compatible version of +no_off.
+no_rot
+ns
+o_alpha
+o_lat_1
+o_lat_2
+o_lat_c
+o_lat_p
+o_lon_1
+o_lon_2
+o_lon_c
+o_lon_p
```

```
+o_proj
+over
+p
+path
+proj      Projection name (see `proj -l`)
+q
+R
+R_a
+R_A       Compute radius such that the area of the sphere is the same as the area of␣
↪the ellipsoid
+rf        Reciprocal of the ellipsoid flattening term (e.g. 298)
+R_g
+R_h
+R_lat_a
+R_lat_g
+rot
+R_V
+s
+south     Denotes southern hemisphere UTM zone
+sym
+t
+theta
+tilt
+to_meter  Multiplier to convert map units to 1.0m
+units     meters, US survey feet, etc.
+vopt
+W
+westo
+x_0       False easting
+y_0       False northing
+zone      UTM zone
```

See GE's libproj4 manual for further details (copy in wayback machine).

Further details for projection at http://www.remotesensing.org/geotiff/proj_list/

Units

Horizontal units can be specified using the +units= keyword with a symbolic name for a unit (ie. us-ft). Alternatively the translation to meters can be specified with the +to_meter keyword (ie. 0.304800609601219 for US feet). The `-lu` argument to cs2cs or proj can be used to list symbolic unit names. The default unit for projected coordinates is the meter. A few special projections deviate from this behaviour, most notably the latlong pseudo-projection that returns degrees.

Vertical Units

Vertical (Z) units can be specified using the `+vunits=` keyword with a symbolic name for a unit (ie. `us-ft`). Alternatively the translation to meters can be specified with the `+vto_meter` keyword (ie. 0.304800609601219 for US feet). The `-lu` argument to cs2cs or proj can be used to list symbolic unit names. If no vertical units are specified, the vertical units will default to be the same as the horizontal coordinates.

Note that vertical unit transformations are only supported in pj_transform() and programs built on that such as cs2cs. The low level projections functions pj_fwd() and pj_inv() and programs using them directly such as proj do not handle vertical units at all.

False Easting/Northing

Virtually all coordinate systems allow for the presence of a false easting (`+x_0`) and northing (`+y_0`). Note that these values are always expressed in meters even if the coordinate system is some other units. Some coordinate systems (such as UTM) have implicit false easting and northing values.

lon_wrap, over - Longitude Wrapping

By default PROJ.4 wraps output longitudes in the range -180 to 180. The +over switch can be used to disable the default wrapping which is done at a low level - in `pj_inv()`. This is particularly useful with projections like eqc where it would desirable for X values past -20000000 (roughly) to continue past -180 instead of wrapping to +180.

The `+lon_wrap` option can be used to provide an alternative means of doing longitude wrapping within `pj_transform()`. The argument to this option is a center longitude. So `+lon_wrap=180` means wrap longitudes in the range 0 to 360. Note that `+over` does **not** disable `+lon_wrap`.

pm - Prime Meridian

A prime meridian may be declared indicating the offset between the prime meridian of the declared coordinate system and that of greenwich. A prime meridian is clared using the "pm" parameter, and may be assigned a symbolic name, or the longitude of the alternative prime meridian relative to greenwich.

Currently prime meridian declarations are only utilized by the `pj_transform()` API call, not the `pj_inv()` and `pj_fwd()` calls. Consequently the user utility `cs2cs` does honour prime meridians but the proj user utility ignores them.

The following predeclared prime meridian names are supported. These can be listed using the cs2cs argument -lm.

```
greenwich 0dE
   lisbon 9d07'54.862"W
    paris 2d20'14.025"E
   bogota 74d04'51.3"E
   madrid 3d41'16.48"W
     rome 12d27'8.4"E
     bern 7d26'22.5"E
  jakarta 106d48'27.79"E
    ferro 17d40'W
 brussels 4d22'4.71"E
stockholm 18d3'29.8"E
   athens 23d42'58.815"E
     oslo 10d43'22.5"E
```

Example of use. The location `long=0`, `lat=0` in the greenwich based lat/long coordinates is translated to lat/long coordinates with Madrid as the prime meridian.

```
cs2cs +proj=latlong +datum=WGS84 +to +proj=latlong +datum=WGS84 +pm=madrid
0 0                             <i>(input)</i>
3d41'16.48"E    0dN 0.000       <i>(output)</i>
```

towgs84 - Datum transformation to WGS84

Datum shifts can be approximated by 3 parameter spatial translations (in geocentric space), or 7 parameter shifts (translation + rotation + scaling). The parameters to describe this can be described using the towgs84 parameter.

In the three parameter case, the three arguments are the translations to the geocentric location in meters.

For instance, the following demonstrates converting from the Greek GGRS87 datum to WGS84.

```
cs2cs +proj=latlong +ellps=GRS80 +towgs84=-199.87,74.79,246.62
    +to +proj=latlong +datum=WGS84
20 35
20d0'5.467"E    35d0'9.575"N 8.570
```

The EPSG database provides this example for transforming from WGS72 to WGS84 using an approximated 7 parameter transformation.

```
cs2cs +proj=latlong +ellps=WGS72 +towgs84=0,0,4.5,0,0,0.554,0.219 \
    +to +proj=latlong +datum=WGS84
4 55
4d0'0.554"E     55d0'0.09"N 3.223
```

The seven parameter case uses `delta_x`, `delta_y`, `delta_z`, `Rx – rotation X`, `Ry – rotation Y`, `Rz – rotation Z`, `M_BF – Scaling`. The three translation parameters are in meters as in the three parameter case. The rotational parameters are in seconds of arc. The scaling is apparently the scale change in parts per million.

A more complete discussion of the 3 and 7 parameter transformations can be found in the EPSG database (trf_method's 9603 and 9606). Within PROJ.4 the following calculations are used to apply the `towgs84` transformation (going to WGS84). The x, y and z coordinates are in geocentric coordinates.

Three parameter transformation (simple offsets):

```
x[io] = x[io] + defn->datum_params[0];
y[io] = y[io] + defn->datum_params[1];
z[io] = z[io] + defn->datum_params[2];
```

Seven parameter transformation (translation, rotation and scaling):

```
#define Dx_BF (defn->datum_params[0])
#define Dy_BF (defn->datum_params[1])
#define Dz_BF (defn->datum_params[2])
#define Rx_BF (defn->datum_params[3])
#define Ry_BF (defn->datum_params[4])
#define Rz_BF (defn->datum_params[5])
#define M_BF  (defn->datum_params[6])

x_out = M_BF*(       x[io] - Rz_BF*y[io] + Ry_BF*z[io]) + Dx_BF;
y_out = M_BF*( Rz_BF*x[io] +       y[io] - Rx_BF*z[io]) + Dy_BF;
z_out = M_BF*(-Ry_BF*x[io] + Rx_BF*y[io] +       z[io]) + Dz_BF;
```

Note that EPSG method 9607 (coordinate frame rotation) coefficients can be converted to EPSG method 9606 (position vector 7-parameter) supported by PROJ.4 by reversing the sign of the rotation vectors. The methods are otherwise the same.

nadgrids - Grid Based Datum Adjustments

In many places (notably North America and Austrialia) national geodetic organizations provide grid shift files for converting between different datums, such as NAD27 to NAD83. These grid shift files include a shift to be applied at each grid location. Actually grid shifts are normally computed based on an interpolation between the containing four grid points.

PROJ.4 currently supports use of grid shift files for shifting between datums and WGS84 under some circumstances. The grid shift table formats are ctable (the binary format produced by the PROJ.4 `nad2bin` program), NTv1 (the old Canadian format), and NTv2 (`.gsb` - the new Canadian and Australian format).

Use of grid shifts is specified using the `nadgrids` keyword in a coordinate system definition. For example:

```
% cs2cs +proj=latlong +ellps=clrk66 +nadgrids=ntv1_can.dat \
    +to +proj=latlong +ellps=GRS80 +datum=NAD83 << EOF
-111 50
EOF
111d0'2.952"W   50d0'0.111"N 0.000
```

In this case the `/usr/local/share/proj/ntv1_can.dat` grid shift file was loaded, and used to get a grid shift value for the selected point.

It is possible to list multiple grid shift files, in which case each will be tried in turn till one is found that contains the point being transformed.

```
cs2cs +proj=latlong +ellps=clrk66 \
          +nadgrids=conus,alaska,hawaii,stgeorge,stlrnc,stpaul \
    +to +proj=latlong +ellps=GRS80 +datum=NAD83 << EOF
-111 44
EOF
111d0'2.788"W   43d59'59.725"N 0.000
```

Skipping Missing Grids

The special prefix `@` may be prefixed to a grid to make it optional. If it not found, the search will continue to the next grid. Normally any grid not found will cause an error. For instance, the following would use the `ntv2_0.gsb` file if available (see [[NonFreeGrids]]), otherwise it would fallback to using the `ntv1_can.dat` file.

```
cs2cs +proj=latlong +ellps=clrk66 +nadgrids=@ntv2_0.gsb,ntv1_can.dat \
    +to +proj=latlong +ellps=GRS80 +datum=NAD83 << EOF
-111 50
EOF
111d0'3.006"W   50d0'0.103"N 0.000
```

The null Grid

A special `null` grid shift file is shift with releases after 4.4.6 (not inclusive). This file provides a zero shift for the whole world. It may be listed at the end of a nadgrids file list if you want a zero shift to be applied to points outside the valid region of all the other grids. Normally if no grid is found that contains the point to be transformed an error will occur.

```
cs2cs +proj=latlong +ellps=clrk66 +nadgrids=conus,null \
    +to +proj=latlong +ellps=GRS80 +datum=NAD83 << EOF
-111 45
EOF
111d0'3.006"W   50d0'0.103"N 0.000

cs2cs +proj=latlong +ellps=clrk66 +nadgrids=conus,null \
    +to +proj=latlong +ellps=GRS80 +datum=NAD83 << EOF
-111 44
-111 55
EOF
111d0'2.788"W   43d59'59.725"N 0.000
111dW   55dN 0.000
```

Downloading and Installing Grids

The source distribution of PROJ.4 contains only the ntv1_can.dat file. To get the set of US grid shift files it is necessary to download an additional distribution of files from the PROJ.4 site, such as ftp://ftp.remotesensing.org/pub/proj/proj-nad27-1.1.tar.gz. Overlay it on the PROJ.4 source distribution, and re-configure, compile and install. The distributed ASCII .lla files are converted into binary (platform specific) files that are installed. On windows using the nmake /f makefile.vc nadshift command in the projsrc directory to build and install these files.

It appears we can't redistribute the Canadian NTv2 grid shift file freely, though it is better than the NTv1 file. However, end users can download it for free from the NRCan web site. After downloading it, just dump it in the data directory with the other installed data files (usually */usr/local/share/proj*). See [[NonFreeGrids]] for details.

Caveats

- Where grids overlap (such as conus and ntv1_can.dat for instance) the first found for a point will be used regardless of whether it is appropriate or not. So, for instance, `` `+nadgrids=ntv1_can.dat` ``,conus would result in the Canadian data being used for some areas in the northern United States even though the conus data is the approved data to use for the area. Careful selection of files and file order is necessary. In some cases border spanning datasets may need to be pre-segmented into Canadian and American points so they can be properly grid shifted
- There are additional grids for shifting between NAD83 and various HPGN versions of the NAD83 datum. Use of these haven't been tried recently so you may encounter problems. The FL.lla, WO.lla, MD.lla, TN.lla and WI.lla are examples of high precision grid shifts. Take care!
- Additional detail on the grid shift being applied can be found by setting the PROJ_DEBUG environment variable to a value. This will result in output to stderr on what grid is used to shift points, the bounds of the various grids loaded and so forth
- PROJ.4 always assumes that grids contain a shift **to** NAD83 (essentially WGS84). Other types of grids might or might not be usable

Axis orientation

Starting in PROJ 4.8.0, the +axis argument can be used to control the axis orientation of the coordinate system. The default orientation is "easting, northing, up" but directions can be flipped, or axes flipped using combinations of the axes in the +axis switch. The values are:

- "e" - Easting
- "w" - Westing
- "n" - Northing
- "s" - Southing
- "u" - Up
- "d" - Down

They can be combined in +axis in forms like:

- `+axis=enu` - the default easting, northing, elevation.
- `+axis=neu` - northing, easting, up - useful for "lat/long" geographic coordinates, or south orientated transverse mercator.
- `+axis=wnu` - westing, northing, up - some planetary coordinate systems have "west positive" coordinate systems

Note that the `+axis` argument only applies to coordinate transformations done through `pj_transform()` (so it works with `cs2cs`, but not with the proj commandline program).

Geodesic Calculations

Contents

Geodesic Calculations

Geodesic calculations are calculations along lines (great circle) on the surface of the earth. They can answer questions like:

- What is the distance between these two points?
- If I travel X meters from point A at bearing phi, where will I be. They are done in native lat-long coordinates, rather than in projected coordinates.

Relevant mailing list threads

- http://thread.gmane.org/gmane.comp.gis.proj-4.devel/3361
- http://thread.gmane.org/gmane.comp.gis.proj-4.devel/3375
- http://thread.gmane.org/gmane.comp.gis.proj-4.devel/3435
- http://thread.gmane.org/gmane.comp.gis.proj-4.devel/3588
- http://thread.gmane.org/gmane.comp.gis.proj-4.devel/3925
- http://thread.gmane.org/gmane.comp.gis.proj-4.devel/4047
- http://thread.gmane.org/gmane.comp.gis.proj-4.devel/4083

Terminology

The shortest distance on the surface of a solid is generally termed a geodesic, be it an ellipsoid of revolution, aposphere, etc. On a sphere, the geodesic is termed a Great Circle.

HOWEVER, when computing the distance between two points using a projected coordinate system, that is a conformal projection such as Transverse Mercator, Oblique Mercator, Normal Mercator, Stereographic, or Lambert Conformal Conic - that then is a GRID distance which can be converted to an equivalent GEODETIC distance using the function for "Scale Factor at a Point." The conversion is then termed "Grid Distance to Geodetic Distance," even though it will not be as exactly correct as a true ellipsoidal geodesic. Closer to the truth with a TM than with a Lambert or other conformal projection, but still not exactly "on."

So, it can be termed "geodetic distance" or a "geodesic distance," depending on just how you got there ...

The Math

Spherical Approximation

The simplest way to compute geodesics is using a sphere as an approximation for the earth. This from Mikael Rittri on the Proj mailing list:

> If 1 percent accuracy is enough, I think you can use spherical formulas with a fixed Earth radius. You can find good formulas in the Aviation Formulary of Ed Williams, http://williams.best.vwh.net/avform.htm.
>
> For the fixed Earth radius, I would choose the average of the:
>
> > c = radius of curvature at the poles, b^2^ / a = radius of curvature in a meridian plane at the equator,
>
> since these are the extreme values for the local radius of curvature of the earth ellipsoid.
>
> If your coordinates are given in WGS84, then
>
> > c = 6 399 593.626 m, b^2^ / a = 6 335 439.327 m,
>
> (see http://home.online.no/~sigurdhu/WGS84_Eng.html) so their average is 6,367,516.477 m. The maximal error for distance calculation should then be less than 0.51 percent.
>
> When computing the azimuth between two points by the spherical formulas, I think the maximal error on WGS84 will be 0.2 degrees, at least if the points are not too far away (less than 1000 km apart, say). The error should be maximal near the equator, for azimuths near northeast etc.

I am not sure about the spherical errors for the forward geodetic problem: point positioning given initial point, distance and azimuth.

Ellipsoidal Approximation

For more accuracy, the earth can be approximated with an ellipsoid, complicating the math somewhat. See the wikipedia page, Geodesics on an ellipsoid, for more information.

Thaddeus Vincenty's method, April 1975

For a very good procedure to calculate inter point distances see:

http://www.ngs.noaa.gov/PC_PROD/Inv_Fwd/ (Fortran code, DOS executables, and an online app)

and algorithm details published in: Vincenty, T. (1975)

Javascript code

Chris Veness has coded Vincenty's formulas as !JavaScript.

distance: http://www.movable-type.co.uk/scripts/latlong-vincenty.html

direct: http://www.movable-type.co.uk/scripts/latlong-vincenty-direct.html

C code

From Gerald Evenden: a library of the converted NGS Vincenty geodesic procedure and an application program, 'geodesic'. In the case of a spherical earth *[Snyder1987]*'s preferred equations are used.

- http://article.gmane.org/gmane.comp.gis.proj-4.devel/3588/

The link in this message is broken. The correct URL is http://home.comcast.net/~gevenden56/proj/

Earlier Mr. Evenden had posted to the PROJ.4 mailing list this code for determination of true distance and respective forward and back azimuths between two points on the ellipsoid. Good for any pair of points that are not antipodal. Later he posted that this was not in fact the translation of NGS FORTRAN code, but something else. But, for what it's worth, here is the posted code (source unknown):

- http://article.gmane.org/gmane.comp.gis.proj-4.devel/3478

PROJ.4 - geod program

The PROJ.4 [wiki:man_geod geod] program can be used for great circle distances on an ellipsoid. As of proj version 4.9.0, this uses a translation of GeographicLib::Geodesic (see below) into C. The underlying geodesic calculation API is exposed as part of the PROJ.4 library (via the geodesic.h header). Prior to version 4.9.0, the algorithm documented here was used: ' Paul D. Thomas, 1970 Spheroidal Geodesics, Reference Systems, and Local Geometry" U.S. Naval Oceanographic Office, p. 162 Engineering Library 526.3 T36s

http://handle.dtic.mil/100.2/AD0703541

GeographicLib::Geodesic

Charles Karney has written a C++ class to do geodesic calculations and a utility GeodSolve to call it. See

- http://geographiclib.sourceforge.net/geod.html

An online version of GeodSolve is available at

- http://geographiclib.sourceforge.net/cgi-bin/GeodSolve

This is an attempt to do geodesic calculations "right", i.e.,

- accurate to round-off (i.e., about 15 nm);
- inverse solution always succeeds (even for near anti-podal points);
- reasonably fast (comparable in speed to Vincenty);
- differential properties of geodesics are computed (these give the scales of geodesic projections);
- the area between a geodesic and the equator is computed (allowing the area of geodesic polygons to be found);
- included also is an implementation in terms of elliptic integrals which can deal with ellipsoids with 0.01 < b/a < 100.

A JavaScript implementation is included, see

- **geo-calc,** a text interface to geodesic calculations;
- **geod-google,** a tool for drawing geodesics on Google Maps.

Implementations in Python, Matlab, C, Fortran , and Java are also available.

The algorithms are described in

- C. F. F. Karney, Algorithms for gedesics, J. Geodesy ‘‘87’’(1), 43-55 (2013), DOI: 10.1007/s00190-012-0578-z; geo-addenda.html.

Triaxial Ellipsoid

A triaxial ellipsoid is a marginally better approximation to the shape of the earth than an ellipsoid of revolution. The problem of geodesics on a triaxial ellipsoid was solved by Jacobi in 1838. For a discussion of this problem see * http://geographiclib.sourceforge.net/html/triaxial.html * the wikipedia entry: Geodesics on a triaxial ellipsoid

The History

The bibliography of papers on the geodesic problem for an ellipsoid is available at

- http://geographiclib.sourceforge.net/geodesic-papers/biblio.html

this includes links to online copies of the papers.

Grids

Contents

- *Brazil*
- *South Africa*
- *Netherlands*

Grid files are important for shifting and transforming between datums

US, Canadian, French and New Zealand

- http://download.osgeo.org/proj/proj-datumgrid-1.5.zip: US, Canadian, French and New Zealand datum shift grids - unzip in the *nad* directory before configuring to add NAD27/NAD83 and NZGD49 datum conversion

Switzerland

Background in ticket #145

We basically have two shift grids available. An official here:

Swiss CHENyx06 dataset in NTv2 format

And a derived in a temporary location which is probably going to disappear soon.

Main problem seems to be there's no mention of distributivity of the grid from the official website. It just tells: "you can use freely". The "contact" link is also broken, but maybe someone could make a phone call to ask for rephrasing that.

HARN

With the support of i-cubed, Frank Warmerdam has written tools to translate the HPGN grids from NOAA/NGS from `.los/.las` format into NTv2 format for convenient use with PROJ.4. This project included implementing a .los/.las reader for GDAL, and an NTv2 reader/writer. Also, a script to do the bulk translation was implemented in https://github.com/OSGeo/gdal/tree/trunk/gdal/swig/python/samples/loslas2ntv2.py. The command to do the translation was:

```
loslas2ntv2.py -auto *hpgn.los
```

As GDAL uses NAD83/WGS84 as a pivot datum, the sense of the HPGN datum shift offsets were negated to map from HPGN to NAD83 instead of the other way. The files can be used with PROJ.4 like this:

```
cs2cs +proj=latlong +datum=NAD83
      +to +proj=latlong +nadgrids=./azhpgn.gsb +ellps=GRS80
```

```
# input:
-112 34
```

```
# output:
111d59'59.996"W 34d0'0.006"N -0.000
```

This was confirmed against the NGS HPGN calculator.

The grids are available at http://download.osgeo.org/proj/hpgn_ntv2.zip

See also:

HTPD describes similar grid shifting

HTDP

HTPD describes the situation with HTDP grids based on NOAA/NGS HTDP Model.

Hungary

Hungarian grid ETRS89 - HD72/EOV (epsg:23700), both horizontal and elevation grids

Non-Free Grids

Not all grid shift files have licensing that allows them to be freely distributed, but can be obtained by users through free and legal methods.

Canada NTv2.0

Although NTv1 grid shifts are provided freely with PROJ.4, the higher-quality NTv2.0 file needs to be downloaded from Natural Resources Canada. More info: http://www.geod.nrcan.gc.ca/tools-outils/ntv2_e.php.

Procedure:

1. Visit the NTv2, and register/login
2. Follow the Download NTv2 link near the bottom of the page.
3. Unzip *ntv2_100325.zip* (or similar), and move the grid shift file *NTV2_0.GSB* to the proj directory (be sure to change the name to lowercase for consistency) * e.g.: *mv NTV2_0.GSB /usr/local/share/proj/ntv2_0.gsb*
4. **Test it using:**

```
cs2cs +proj=latlong +ellps=clrk66 +nadgrids=@ntv2_0.gsb +to +proj=latlong␣
↪+ellps=GRS80 +datum=NAD83
-111 50
```

```
111d0'3.006"W   50d0'0.103"N 0.000  # correct answer
```

Australia

Geocentric Datum of Australia AGD66/AGD84

Canada

Canadian NTv2 grid shift binary for NAD27 <=> NAD83.

Germany

German BeTA2007 DHDN GK3 => ETRS89/UTM

Great Britain

Great Britain's OSTN15_NTv2: OSGB 1936 => ETRS89

Great Britain's OSTN02_NTv2: OSGB 1936 => ETRS89

Austria

Austrian Grid for MGI

Spain

Spanish grids for ED50.

Portugal

Portuguese grids for ED50, Lisbon 1890, Lisbon 1937 and Datum 73

Brazil

Brazilian grids for datums Corrego Alegre 1961, Corrego Alegre 1970-72, SAD69 and SAD69(96)

South Africa

South African grid (Cape to Hartebeesthoek94 or WGS84)

Netherlands

Dutch grid (Registration required before download)

HTPD

Contents

This page documents use of the *crs2crs2grid.py* script and the HTDP (Horizontal Time Dependent Positioning) grid shift modelling program from NGS/NOAA to produce PROJ.4 compatible grid shift files for fine grade conversions between various NAD83 epochs and WGS84. Traditionally PROJ.4 has treated NAD83 and WGS84 as equivalent and failed to distinguish between different epochs or realizations of those datums. At the scales of much mapping this is adequate but as interest grows in high resolution imagery and other high resolution mapping this is inadequate. Also, as the North American crust drifts over time the displacement between NAD83 and WGS84 grows (more than one foot over the last two decades).

Getting and building HTDP

The HTDP modelling program is in written FORTRAN. The source and documentation can be found on the HTDP page at http://www.ngs.noaa.gov/TOOLS/Htdp/Htdp.shtml

On linux systems it will be necessary to install *gfortran* or some FORTRAN compiler. For ubuntu something like the following should work.

```
apt-get install gfortran
```

To compile the program do something like the following to produce the binary "htdp" from the source code.

```
gfortran htdp.for -o htdp
```

Getting crs2crs2grid.py

The *crs2crs2grid.py* script can be found at https://github.com/OSGeo/gdal/tree/trunk/gdal/swig/python/samples/crs2crs2grid.py

It depends on having the GDAL Python bindings operational. If they are not available you will get an error something like the following:

```
Traceback (most recent call last):
  File "./crs2crs2grid.py", line 37, in <module>
    from osgeo import gdal, gdal_array, osr
ImportError: No module named osgeo
```

Usage

```
   crs2crs2grid.py
            <src_crs_id> <src_crs_date> <dst_crs_id> <dst_crs_year>
            [-griddef <ul_lon> <ul_lat> <ll_lon> <ll_lat> <lon_count> <lat_count>]
            [-htdp <path_to_exe>] [-wrkdir <dirpath>] [-kwf]
            -o <output_grid_name>

-griddef: by default the following values for roughly the continental USA
          at a six minute step size are used:
          -127 50 -66 25 251 611
-kwf: keep working files in the working directory for review.
```

```
crs2crs2grid.py 29 2002.0 8 2002.0 -o nad83_2002.ct2
```

The goal of *crs2crs2grid.py* is to produce a grid shift file for a designated region. The region is defined using the *-griddef* switch. When missing a continental US region is used. The script creates a set of sample points for the grid definition, runs the "htdp" program against it and then parses the resulting points and computes a point by point shift to encode into the final grid shift file. By default it is assumed the *htdp* program will be in the executable path. If not, please provide the path to the executable using the *-htdp* switch.

The *htdp* program supports transformations between many CRSes and for each (or most?) of them you need to provide a date at which the CRS is fixed. The full set of CRS Ids available in the HTDP program are:

```
 1...NAD_83(2011) (North America tectonic plate fixed)
 29...NAD_83(CORS96)   (NAD_83(2011) will be used)
 30...NAD_83(2007)     (NAD_83(2011) will be used)
```

```
 2...NAD_83(PA11) (Pacific tectonic plate fixed)
 31...NAD_83(PACP00)  (NAD 83(PA11) will be used)
 3...NAD_83(MA11) (Mariana tectonic plate fixed)
 32...NAD_83(MARP00)  (NAD_83(MA11) will be used)

 4...WGS_72                              16...ITRF92
 5...WGS_84(transit) = NAD_83(2011)      17...ITRF93
 6...WGS_84(G730) = ITRF92               18...ITRF94 = ITRF96
 7...WGS_84(G873) = ITRF96               19...ITRF96
 8...WGS_84(G1150) = ITRF2000            20...ITRF97
 9...PNEOS_90 = ITRF90                   21...IGS97 = ITRF97
10...NEOS_90 = ITRF90                    22...ITRF2000
11...SIO/MIT_92 = ITRF91                 23...IGS00 = ITRF2000
12...ITRF88                              24...IGb00 = ITRF2000
13...ITRF89                              25...ITRF2005
14...ITRF90                              26...IGS05 = ITRF2005
15...ITRF91                              27...ITRF2008
                                         28...IGS08 = ITRF2008
```

The typical use case is mapping from NAD83 on a particular date to WGS84 on some date. In this case the source CRS Id "29" (NAD_83(CORS96)) and the destination CRS Id is "8 (WGS_84(G1150)). It is also necessary to select the source and destination date (epoch). For example:

```
crs2crs2grid.py 29 2002.0 8 2002.0 -o nad83_2002.ct2
```

The output is a CTable2 format grid shift file suitable for use with PROJ.4 (4.8.0 or newer). It might be utilized something like:

```
cs2cs +proj=latlong +ellps=GRS80 +nadgrids=./nad83_2002.ct2 +to +proj=latlong␣
↪+datum=WGS84
```

See Also

- http://www.ngs.noaa.gov/TOOLS/Htdp/Htdp.shtml - NGS/NOAA page about the HTDP model and program. Source for the HTDP program can be downloaded from here.

Development

These pages are primarily focused towards developers either contributing to the proj.4 project or using the library in their own software.

API

Contents

Introduction

Procedure `pj_init()` selects and initializes a cartographic projection with its argument control parameters. `argc` is the number of elements in the array of control strings argv that each contain individual cartographic control keyword assignments (+ proj arguments). The list must contain at least the proj=projection and Earth's radius or elliptical parameters. If the initialization of the projection is successful a valid address is returned otherwise a NULL value.

The `pj_init_plus()` function operates similarly to `pj_init()` but takes a single string containing the definition, with each parameter prefixed with a plus sign. For example `+proj=utm +zone=11 +ellps=WGS84`.

Once initialization is performed either forward or inverse projections can be performed with the returned value of `pj_init()` used as the argument proj. The argument structure projUV values u and v contain respective longitude and latitude or x and y. Latitude and longitude are in radians. If a projection operation fails, both elements of projUV are set to `HUGE_VAL` (defined in `math.h`).

Note: all projections have a forward mode, but some do not have an inverse projection. If the projection does not have an inverse the projPJ structure element inv will be `NULL`.

The `pj_transform` function may be used to transform points between the two provided coordinate systems. In addition to converting between cartographic projection coordinates and geographic coordinates, this function also takes care of datum shifts if possible between the source and destination coordinate system. Unlike `pj_fwd()` and `pj_inv()` it is also allowable for the coordinate system definitions `(projPJ *)` to be geographic coordinate systems (defined as `+proj=latlong`). The x, y and z arrays contain the input values of the points, and are replaced with the output values. The function returns zero on success, or the error number (also in `pj_errno`) on failure.

Memory associated with the projection may be freed with `pj_free()`.

Example

The following program reads latitude and longitude values in decimal degress, performs Mercator projection with a Clarke 1866 ellipsoid and a 33° latitude of true scale and prints the projected cartesian values in meters:

```
#include <proj_api.h>

main(int argc, char **argv) {
     projPJ pj_merc, pj_latlong;
     double x, y;

     if (!(pj_merc = pj_init_plus("+proj=merc +ellps=clrk66 +lat_ts=33")) )
        exit(1);
     if (!(pj_latlong = pj_init_plus("+proj=latlong +ellps=clrk66")) )
        exit(1);
     while (scanf("%lf %lf", &x, &y) == 2) {
        x *= DEG_TO_RAD;
        y *= DEG_TO_RAD;
        p = pj_transform(pj_latlong, pj_merc, 1, 1, &x, &y, NULL );
        printf("%.2f\t%.2f\n", x, y);
     }
     exit(0);
}
```

For this program, an input of `-16 20.25` would give a result of `-1495284.21 1920596.79`.

API Functions

pj_transform

```
int pj_transform( projPJ srcdefn,
                  projPJ dstdefn,
                  long point_count,
                  int point_offset,
                  double *x,
                  double *y,
                  double *z );
```

Transform the x/y/z points from the source coordinate system to the destination coordinate system.

`srcdefn`: source (input) coordinate system.

`dstdefn`: destination (output) coordinate system.

`point_count`: the number of points to be processed (the size of the x/y/z arrays).

`point_offset`: the step size from value to value (measured in doubles) within the x/y/z arrays - normally 1 for a packed array. May be used to operate on xyz interleaved point arrays.

`x`/`y`/`z`: The array of X, Y and Z coordinate values passed as input, and modified in place for output. The Z may optionally be NULL.

`return`: The return is zero on success, or a PROJ.4 error code.

The `pj_transform()` function transforms the passed in list of points from the source coordinate system to the destination coordinate system. Note that geographic locations need to be passed in radians, not decimal degrees, and will be returned similarly. The `z` array may be passed as NULL if Z values are not available.

If there is an overall failure, an error code will be returned from the function. If individual points fail to transform - for instance due to being over the horizon - then those x/y/z values will be set to `HUGE_VAL` on return. Input values that are `HUGE_VAL` will not be transformed.

pj_init_plus

```
projPJ pj_init_plus(const char *definition);
```

This function converts a string representation of a coordinate system definition into a projPJ object suitable for use with other API functions. On failure the function will return NULL and set pj_errno. The definition should be of the general form `+proj=tmerc +lon_0 +datum=WGS84`. Refer to PROJ.4 documentation and the *Parameters* notes for additional detail.

Coordinate system objects allocated with `pj_init_plus()` should be deallocated with `pj_free()`.

pj_free

```
void pj_free( projPJ pj );
```

Frees all resources associated with pj.

pj_is_latlong

```
int pj_is_latlong( projPJ pj );
```

Returns TRUE if the passed coordinate system is geographic (`proj=latlong`).

pj_is_geocent

```
int pj_is_geocent( projPJ pj );``
```

Returns TRUE if the coordinate system is geocentric (`proj=geocent`).

pj_get_def

```
char *pj_get_def( projPJ pj, int options);``
```

Returns the PROJ.4 initialization string suitable for use with `pj_init_plus()` that would produce this coordinate system, but with the definition expanded as much as possible (for instance `+init=` and `+datum=` definitions).

pj_latlong_from_proj

```
projPJ pj_latlong_from_proj( projPJ pj_in );``
```

Returns a new coordinate system definition which is the geographic coordinate (lat/long) system underlying `pj_in`.

pj_set_finder

```
void pj_set_finder( const char *(*new_finder)(const char *) );``
```

Install a custom function for finding init and grid shift files.

pj_set_searchpath

```
void pj_set_searchpath ( int count, const char **path );``
```

Set a list of directories to search for init and grid shift files.

pj_deallocate_grids

```
void pj_deallocate_grids( void );``
```

Frees all resources associated with loaded and cached datum shift grids.

pj_strerrno

```
char *pj_strerrno( int );``
```

Returns the error text associated with the passed in error code.

pj_get_errno_ref

```
int *pj_get_errno_ref( void );``
```

Returns a pointer to the global pj_errno error variable.

pj_get_release

```
const char *pj_get_release( void );``
```

Returns an internal string describing the release version.

Obsolete Functions

```
XY pj_fwd( LP lp, PJ *P );
LP pj_inv( XY xy, PJ *P );
projPJ pj_init(int argc, char **argv);
```

Threads

Contents

- *src/multistresstest.c*

This page is about efforts to make PROJ.4 thread safe.

Key Thread Safety Issues

- the global pj_errno variable is shared between threads and makes it essentially impossible to handle errors safely. Being addressed with the introduction of the projCtx execution context.
- the datum shift using grid files uses globally shared lists of loaded grid information. Access to this has been made safe in 4.7.0 with the introduction of a proj.4 mutex used to protect access to these memory structures (see pj_mutex.c).

projCtx

Primarily in order to avoid having pj_errno as a global variable, a "thread context" structure has been introduced into a variation of the PROJ.4 API for the 4.8.0 release. The pj_init() and pj_init_plus() functions now have context variations called pj_init_ctx() and pj_init_plus_ctx() which take a projections context.

The projections context can be created with pj_ctx_alloc(), and there is a global default context used when one is not provided by the application. There is a pj_ctx_ set of functions to create, manipulate, query, and destroy contexts. The contexts are also used now to handle setting debugging mode, and to hold an error reporting function for textual error and debug messages. The API looks like:

```
projPJ pj_init_ctx( projCtx, int, char ** );
projPJ pj_init_plus_ctx( projCtx, const char * );

projCtx pj_get_default_ctx(void);
projCtx pj_get_ctx( projPJ );
void pj_set_ctx( projPJ, projCtx );
projCtx pj_ctx_alloc(void);
void    pj_ctx_free( projCtx );
int pj_ctx_get_errno( projCtx );
void pj_ctx_set_errno( projCtx, int );
void pj_ctx_set_debug( projCtx, int );
void pj_ctx_set_logger( projCtx, void (*)(void *, int, const char *) );
void pj_ctx_set_app_data( projCtx, void * );
void *pj_ctx_get_app_data( projCtx );
```

Multithreaded applications are now expected to create a projCtx per thread using pj_ctx_alloc(). The context's error handlers, and app data may be modified if desired, but at the very least each context has an internal error value accessed with pj_ctx_get_errno() as opposed to looking at pj_errno.

Note that pj_errno continues to exist, and it is set by pj_ctx_set_errno() (as well as setting the context specific error number), but pj_errno still suffers from the global shared problem between threads and should not be used by multithreaded applications.

Note that pj_init_ctx(), and pj_init_plus_ctx() will assign the projCtx to the created projPJ object. Functions like pj_transform(), pj_fwd() and pj_inv() will use the context of the projPJ for error reporting.

src/multistresstest.c

A small multi-threaded test program has been written (src/multistresstest.c) for testing multithreaded use of PROJ.4. It performs a series of reprojections to setup a table expected results, and then it does them many times in several threads

to confirm that the results are consistent. At this time this program is not part of the builds but it can be built on linux like:

```
gcc -g multistresstest.c .libs/libproj.so -lpthread -o multistresstest
./multistresstest
```

Language bindings

Proj.4 bindings are available for a number of different development platforms.

Python

pyproj: Python interface (wrapper for proj.4)

Ruby

proj4rb: Bindings for proj.4 in ruby

TCL

proj4tcl: Bindings for proj.4 in tcl (critcl source)

MySQL

fProj4: Bindings for proj.4 in MySQL

Excel

proj.xll: Excel add-in for proj.4 map projections

Visual Basic

PROJ.4 VB Wrappers: By Eric G. Miller.

Glossary

Pseudocylindrical Projection Pseudocylindrical projections have the mathematical characteristics of

$$x = f(\lambda, \phi)$$
$$y = g(\phi)$$

where the parallels of latitude are straight lines, like cylindrical projections, but the meridians are curved toward the center as they depart from the equator. This is an effort to minimize the distortion of the polar regions inherent in the cylindrical projections.

Pseudocylindrical projections are almost exclusively used for small scale global displays and, except for the Sinusoidal projection, only derived for a spherical Earth. Because of the basic definition none of the pseudocylindrical projections are conformal but many are equal area.

To further reduce distortion, pseudocylindrical are often presented in interrupted form that are made by joining several regions with appropriate central meridians and false easting and clipping boundaries. Interrupted Homolosine constructions are suited for showing respective global land and oceanic regions, for example. To reduce the lateral size of the map, some uses remove an irregular, North-South strip of the mid-Atlantic region so that the western tip of Africa is plotted north of the eastern tip of South America.

License

Author Frank Warmerdam

Contact warmerdam@pobox.com

Date 2001

PROJ.4 has been placed under an MIT license. I believe this to be as close as possible to public domain while satisfying those who say that a copyright notice is required in some countries. The COPYING file read as follows:

All source, data files and other contents of the PROJ.4 package are available under the following terms. Note that the PROJ 4.3 and earlier was "public domain" as is common with US government work, but apparently this is not a well defined legal term in many countries. I am placing everything under the following MIT style license because I believe it is effectively the same as public domain, allowing anyone to use the code as they wish, including making proprietary derivatives.

Though I have put my own name as copyright holder, I don't mean to imply I did the work. Essentially all work was done by Gerald Evenden.

```
Copyright (c) 2000, Frank Warmerdam

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

References

CHAPTER

TWO

MAILING LIST

The Proj.4 mailing list can be found at http://lists.maptools.org/mailman/listinfo/proj

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

BIBLIOGRAPHY

[Evenden1995] Evenden, G. I., 1995, Cartograpic Projection Procedures for the UNIX Environment - A User's Manual

[Evenden2005] Evenden, G. I., 2005, libproj4: A Comprehensive Library of Cartographic Projection Functions (Preliminary Draft)

[Steers1970] Steers, J.A., 1970, An introduction to the study of map projections (15th ed.): London, Univ. London Press, p. 229

[Snyder1987] Snyder. John P. 1987. Map Projections - A Working Manual. US. Geological Survey professional paper; 1395.

[Snyder1993] Snyder, 1993, Flattening the Earth, Chicago and London, The university of Chicago press

[EberHewitt1979] Eber, L.E., and R.P. Hewitt. 1979. Conversion algorithms for the CALCOFI station grid. California Cooperative Oceanic Fisheries Investigations Reports 20:135-137.

[WeberMoore2013] Weber, E.D., and T.J. Moore. 2013. Corrected Conversion Algorithms For The Calcofi Station Grid And Their Implementation In Several Computer Languages. California Cooperative Oceanic Fisheries Investigations Reports 54.

[CalabrettaGreisen2002] 13. Calabretta and E. Greisen, 2002, "Representations of celestial coordinates in FITS". Astronomy & Astrophysics 395, 3, 1077–1122.

[ChanONeil1975] 6. Chan and E.M.O'Neill, 1975, "Feasibility Study of a Quadrilateralized Spherical Cube Earth Data Base". Tech. Rep. EPRF 2-75 (CSC), Environmental Prediction Research Facility.

[ONeilLaubscher1976] E.M. O'Neill and R.E. Laubscher, 1976, "Extended Studies of a Quadrilateralized Spherical Cube Earth Data Base". Tech. Rep. NEPRF 3-76 (CSC), Naval Environmental Prediction Research Facility.

[LambersKolb2012] 13. Lambers and A. Kolb, 2012, "Ellipsoidal Cube Maps for Accurate Rendering of Planetary-Scale Terrain Data", Proc. Pacfic Graphics (Short Papers).

INDEX