

## NAME

processGPX

## SYNOPSIS

`processGPX` [options] <input files>

## VERSION

0.60

## DESCRIPTION

processGPX is a series of algorithms to improve and create GPX files, in particular for cycling emulation platforms like BikeTerra. It was originally written for RGT.

At the time of this writing, BikeTerra allows for the creation of custom routes whereby a GPX file can be submitted for conversion into a virtual environment. The GPX file can be initially generate by on-line mapping tools, among which one excellent option is Strava Route Editor, which exports GPX. However, these tools tend to produce GPX files with too low resolution in position, and small errors in altitude, so that corners are not sufficiently round, and the gradient between points can have anomalously large magnitude.

Additionally, BikeTerra requires the route to be specified continuously from start to finish, rather than allowing arbitrary navigation over a network of roads, and so sometimes the same section of road needs to be repeated, either in the same or in the opposite direction. If you repeat sections of roadway, make sure there is no overlap or road crossings, as as of this writing BikeTerra does not produce good results with these. Note point-to-point courses in BikeTerra automatically feature turn-around loops at each end, so there is no need to do anything to allow returning on a point-to-point route.

These are just some examples of the sort of processing which can be done to improve the quality and functionality of BikeTerra and presumably other cycling emulators as well. There exist online tools such as GPX Magic, which is excellent, but requires extensive user interaction, and a command-line tool able to process an entire file at once has benefit. A nice approach is to general route editing with GPXMagic, then use `processGPX` to process the result.

The file takes GPX files on the command line, and generates a file with a suffix "`_processed`" for each, unless the `-out` option is used, in which case that is used as the output (this works for only one input file). So for example, if I type:

```
processGPX myFavoriteRoute.gpx
```

the result will be a file:

*myFavoriteRoute\_processed.gpx*

This file will be essentially equivalent to the input file, in the absence of any comment line options, although if there are any "zig-zags" identified, those will be removed. The file will also be checked for "loops", where the direction spins around within 100 meters, which might be from a poorly placed control point with mapping software.

If an alternate filename were desired for the output, that could be specified with the `-out` option:

```
processGPX myFavoriteRoute.gpx -out myFavoriteRouteCopy.gpx
```

where the order of command line options does not matter, except that the same option listed more than once will result in parameters specified last being used.

Typically GPX file contain a single "track", each of which contains multiple "segments", each of which contains multiple connected "points". Segments are connected as well as points within the segments. However, GPX file may also contain multiple tracks. processGPX is set up to process one track at a time within a GPX file. By default, this is the first track, or track #1 counting 1, 2, 3 ... If you want to specify a track other than 1, then use either the `-track` option (followed by a number 1, 2, 3...), or

put the number after a ":" at the end of a filename. So for example, if I wanted track #2 in a file `file.gpx`, I could specify either `-track 2`, or `file.gpx:2`. For the `-join` option only the filename suffix works, so `-join file.gpx:2`.

The program will calculate a quality of both the original GPX file, and of the result of processing, which is based on how much the grade changes point-to-point and also how much the direction changes point-to-point. The goal is to only have abrupt gradient changes where they actually exist in the real-life course, and to smooth corners so the direction does not abruptly change. This quality score can provide guidance to if altitude smoothing and point density are sufficient.

It also reports the length of the course, calculated treating the earth as a sphere of a given radius. The calculation in BikeTerra may differ slightly, so distances may differ by a few meters.

There are a number of options, but a key one is `-auto`. It is designed to generate a decent result in a large fraction of the cases. If you only use this option, you should still be happy with results. Other options, however, allow for fine-tuning of the amount of route processing. They can be used in conjunction with `-auto` to override those settings.

One important difference is how point-to-point routes and lappable routes are treated. You can specify a loop course as `-lap`. If you want the code to automatically determine whether a course is a point-to-point or lappable, use `-autoLap`. This option is implied with the `-auto` option.

There are a number of options to create "out-and-back" routes. These were required by RGT. However, BikeTerra automatically puts turn-around loops at each end of a point-to-point course. So you do not need use these options with BikeTerra. These options are thus deprecated.

## DEPENDENCIES

This code uses the following Perl modules, which must be installed, for example with the "cpan" command-line tool:

```
Getopt::Long : used for processing command-line options
Geo::Gpx     : parsing and generating GPX files
XML::Descent : processing XML (required by Geo::Gpx, as well)
POSIX        : floor function
Date::Parse  : for parsing time, with the -startTime option
Pod::Usage   : for the -help option
```

For processing BikeTerra routes or GPX files directly you need: `HTTP::Tiny` : used for downloading data from the web.

and for BikeTerra route data (as opposed to GPX files) you need: `JSON` : used for parsing JSON-formatted files.

## SPECIFYING INPUT SOURCES

There are a number of ways to specify input sources.

The most common way is to provide an input filename for a GPX file. So, for example, if you have a file `myFavoriteRoute.gpx`, you'd list that as a commane-line element.

Optionally a track number can be specified, if the default of using the first (or only) track isn't wanted. So `myFavoriteRoute.gpx:2` would use the second track in the file.

For BikeTerra users, routes can be downloaded from BikeTerra directly and used as input. There's two sources. One is to use the GPX file which was sent to Biketerra. BikeTerra takes other data formats, like FIT, but this only works here with GPX files. This is specified with `BikeTerraRoute:` followed by the route number. So for example, `BikeTerraRoute:10605` would use the route associated with BikeTerra route 10605. Again, the track number can be appended, so if `BikeTerraRoute:10605:1` specified the first track in the GPX file associated with route 10605. Typically these files will have only a single track, so this is not an issue.

An alternate approach with BikeTerra routes is to download a GPX file associated with the present course. BikeTerra simplifies route data on upload, and the points may be modified via the route editor. This option will include any such simplifications and edits. It is specified with `BikeTerraRoute:` followed by the route number. You can additionally specify a track number but there will only be one track (as of this writing: maybe that will change at some point).

You can shortcut these as `BTGPX:` and `BTRoute:`, for example. Generally the "Bike" can be shortened to B, and/or the "Terra" to T. Case does not matter.

Another option is to omit a filename, or to specify an empty dash `-`. This indicates that the standard input is to be used, for example with a pipe. As with filenames, a track number can be appended to the dash, for example `- : 2` would pick the second track from the GPX provided as standard input.

## OPTIONS

The program is generally invoked:

```
processGPX [options] <inputfilename> ...
```

where multiple input file names may be specified, and zero or more options may be specified.

Options are case-insensitive and come in three varieties:

**flags:** specifying the option by itself invokes the option. For example, `-splineFit` specifies that spline fitting is enabled. Flag options can be negated with the same option but with "no" preceding, for example, `-noSplineFit` disables spline fitting.

**values:** the option specification is followed by a value. This may simultaneously invoke an option. For example, `-spacing 10` sets the spacing for point interpolation to 10 meters, and additionally turns on point interpolation.

**lists:** Some options allow for multiple values, for example `-uniformGradient` or `-autoSegments`. See details in the description of the specific option.

The options are the following, excluding negations, which for flags are the option with `no` preceding the keyword.

### **-addCurvature**

This option adds a "curvature" extension field (in inverse meters) to the GPX file. Curvature is calculated as the ratio of the rate of change of angle with distance. Note for a unit circle (radius = 1) the rate of change of angle (in radians) with position (along the perimeter) is 1. In general this is the reciprocal of the circle radius.

### **-addDirection**

This option adds an extensions "heading" field relative to east (in degrees) to the GPX file.

examples:

0 degrees: east

90 degrees: north

180 degrees: west

270 degrees south

Note this is not "heading", as it is relative to east.

Direction is calculated for each point as the average of the directions ahead and behind. The direction of the first or last points in a point-to-point course are calculated in only one direction. Directions for loop courses are calculated assuming the loop. The direction never changes by more than +/- 180 degrees from one point to the next, so the direction has no limits: it can be positive, negative, and outside the range from -360 to +360 degrees. For example, if a route started east (0) then lapped around counterclockwise 10 times, the final direction would be approximately 3600 degrees.

**-addCornerWaypoints**

This option places waypoints at positions of relatively high curvature, marking them as corners. They can be viewed, for example, in *GPXStudio*. The future intent is for them to be useful in a game such as *BikeTerra* for generating roadside signs.

It is likely this option will need to be tuned with `-curvatureThreshold`, and possibly `-curvatureSmoothing`, depending on the route.

**-addGradient**

This option adds a forward "gradient" extensions field, which is calculated for each point as the ratio of the altitude change to the horizontal distance to the next point.

**-addGradientSigns**

This flag is an option to calculate regions of exceptional gradient and place "gradient signs". At present this is equivalent to `-addGradientWaypoints`, although in future versions it may be changed to support a feature on a virtual game such as *BikeTerra*.

**-addGradientWaypoints**

This flag is an option to calculate regions of exceptional gradient and place GPX waypoints. These are visible, for example, with *GPXStudio*. Waypoints are tagged depending on if they are associated with the forward or reverse direction of travel.

**-addHeading**

Alias for `-addDirection`.

**-addSigma**

This writes the autosmoothing "sigma" to the GPX file as an extensions field, but only if `-autosmoothingZ` is positive. It is useful for tuning the autosmoothing scale length.

**-align <option>**

Synonym for `-snap`

**-alignAltitude <meters>**

Synonym for `-snapAltitudes`

**-alignDistance <meters>**

Synonym for `-snapDistance`

**-alignTransition <meters>**

Synonym for `-snapTransition`

**-alignZ <meters>**

This is a synonym for `-snapAltitude`.

**-anchorSF**

For point-to-point routes, this specifies that the start and finish points should be anchored. Smoothing would otherwise cause these points to shift somewhat.

**-append <meters>**

A distance in meters which should be added to point-to-point courses. In RGT, the finish line was typically placed 140 meters prior to the end of a point-to-point course, so if for example you design a course to be exactly 10 miles, (a typical UK time trial distance), then to put the finish line at the end of those 10 miles, the option `"-append 140"` would add 140 meters. This buffer was needed by RGT because riders group along the road-side after crossing the finish. Note the start line would also need to be extended, via `-prepend`, by 60 meters, in the case of RGT.

Extended points will be set to the altitude of the final point.

Since BikeTerra tends to create loop courses when start and finish points are close, the code may put a bend in the road it creates with this command, if both `-append` and `-prepend` are specified (perhaps via `-extend`).

Negative values crop the course by the negative distance. So `"-append -100"` will move the finish line back 100 meters. This was useful for preventing RGT from connecting the finish and start points, which has a 100 meter threshold, as this is written.

**-arcFitAngle <degrees>**

**-arcFitDeps <degrees>**

If arc fits are desired, specifying this will cause arc fit interpolation to be done for corners turning at least this much, but less than the `-arcFitMaxDeps` option. A typical value is 5 degrees. This angle is the angle between interpolated course points

For a given 4 points *a*, *b*, *c*, and *d*, it interpolates between points *b* and *c* by fitting a circle through points *a*, *b*, and *c*, fitting a second circle between points *b*, *c*, and *d*, and then doing a weighted average of these two fits based on the relative distance of the interpolated point from points *b* and *c*. If starting with points forming the vertices of a polygon, for example, the resulting interpolation will yield a circular course, which may or may not be what you want.

This algorithm tends to result in more sweeping corners than spline interpolation, which may or may not reflect reality. Also if there are any anomalous points then there's the chance this algorithm will exaggerate these anomalies. It's recommended you very carefully check the results of applying this option. It's perhaps best for circuits with sweeping corners rather than straights connected with tighter turns.

See also `-arcFitMaxDeps`.

**-arcFitEnd <meters>**

Where to end arc fitting, if any. The default is to end any arc fitting at the end of the course.

**-arcFitMaxAngle <degrees>**

**-arcFitMaxDeps <degrees>**

If a `-arcFitDeps` option is specified, specifying this limit the maximum angle corner for which arc fit interpolation will be applied. ArcFits are good for gradual, rounded corners but are not good for sharp corners, so an upper bound in the 60 degree range (which is the default) works generally well. ArcFits have the advantage of rounding corners without "blunting" them, but sometimes they create "S" shapes where they are not wanted. Make sure to check the results if using arc fit interpolation: strange results can occur if the corner is too sharp.

Arc fitting is a variation on spline fitting where corners are fit to circles rather than splines. If this option is set too large then there's a risk of getting dramatically incorrect trajectories for sharp corners. See also the notes in `-arcFitDeps`.

**-arcFitStart <meters>**

Where to start arc fitting, if any. The default is to start any arc fitting at the start of the course. This can be after `-arcFitEnd` and the region will wrap-around.

**-author <string>**

Provide the author of the GPX, as a string. The default will preserve the author of the source GPX.

**-auto**

This option automatically turns on options based on "best practices". It will not turn anything off, so can be used in conjunction with other options. Options specifically listed over-ride the `-auto` option. So for example, to specify that a course is point-to-point and to skip auto-loop detection normally done with `-auto`, you can do:

`-processGPX -nolop -auto`

**-autoLap**

---

Synonym for `-autoLoop`

**`-autoLoop`**

Automatically determine if the `-loop` option should be invoked based on the relative position and orientation of the beginning and end of the GPX track.

**`-autoSegments <threshold> <optional power>`**

Whether to automatically generate segments for the climb. The threshold is the vertical meters of a 10% gradient climb, where climbs at alternate gradients are adjusted by a term  $(\text{gradient} / 10\%)^{\text{power}}$ , where the default power is 0.5. The names can be set with "autoSegmentNames", with a generic default.

This was designed for use in RGT and has no application in any present program other than GPXMagic.

**`-autoSegmentFinishMargin <meters>`**

A minimum buffer between an auto-generated segment and the finish banner. This defaults to 20 meters to allow room for riders to see a finish banner after passing the segment banner.

This was designed for use in RGT and has no application in any present program other than GPXMagic.

**`-autoSegmentMargin <meters>`**

A minimum buffer between auto-generated segments. The default is 400 meters, based on experiments with creating magic roads with different gaps.

This was designed for use in RGT and has no application in any present program other than GPXMagic.

**`-autoSegmentNames <list of names>`**

A list of names for automatically generated segments, separated by commas or semicolons. The default is "climb" followed by a number. If there are more segments than names, the default will be used for the additional climbs.

This was designed for use in RGT and has no application in any present program other than GPXMagic.

**`-autoSegmentPower <number>`**

If the gradient power for autosegments is not provided in the `-autoSegments` option as a second value, then use this value. The default is 0.5. Decent numbers are from 0.25 to 3, depending on how much focus you want to put on steepness in defining and rating climbs.

This was designed for use in RGT and has no application in any present program other than GPXMagic.

**`-autoSegmentStartMargin <meters>`**

A minimum buffer between the start banner and the first segment. This defaults to 340 meters, based on experiments. Note this results in a 400 meter distance from the start of the GPX file.

This was designed for use in RGT and has no application in any present program other than GPXMagic.

**`-autoSegmentStretch <relative amount>`**

The amount of distance increase one is willing to extend an auto-segment to reach a true peak or valley. This is applied to each side, so a value of 0.05 will result in up to a 10% increase in the total length of an automatically generated segment. The increase will not be applied if it would result in the loss of mandated margin (by `-autoSegmentMargin`) to an adjacent segment, or too close to the start (`-autoSegmentStartMargin`) or finish (`-autoSegmentFinishMargin`)

This was designed for use in RGT and has no application in any present program other than GPXMagic.

**-autoSmoothG <scale factor>**

This invokes gradient autosmoothing selectively along the course, using a similar syntax to `-autoSmooth` and `-autoSmoothZ`. For details on gradient smoothing, see the `-smoothG` option.

**-autoSmoothZ <scale factor>**

This invokes an auto-smoothing algorithm whereby more altitude smoothing is applied where gradient changes rapidly point-to-point, less altitude smoothing where gradient changes slowly. This is done after the conventional position and altitude smoothing specified with `-smooth` and/or `-smoothZ`. So it allows the use of less of these fixed smoothings, and apply an additional smoothing where it is most needed. An application of this would be a course where the altitude is low quality on a subset of the total course, and one thus wants more averaging averaging focused on that subset.

The scale factor controls how much smoothing is applied. It is roughly calibrated so "1" works well, but you can try reducing that to 0.5, for example, and see if the results are still satisfactory, or increasing it to 2 if the gradient is still too spiky.

Consider using `-smoothZ` instead unless you think there's a reason the quality of the data is worse in some areas than in others: that applies uniform smoothing over the whole course. Or the two can be combined.

**-autoSpacing**

Specify that points will be interpolated based on an algorithm. The key parameter is "smoothAngle", which determines where points are placed. This only makes sense if the a smoothing distance or a minRadius is also provided, via the `-smooth` or `-minRadius` options.

This option is highly recommended (either explicitly or implicitly via `-auto` or `-outAndBack`): it greatly improves the smoothness of corners.

**-autoSplits <number>**

Number of files into which the original should be automatically split. So 1 here will do nothing. 2 would divide the original into two equal length routes.

These splits are in addition to splits generated by the `-splitAt` option. If you want to precisely place the split points, then the `-splitAt` option is better.

If you specify an output file and do not specify a specific split number with the `-splitNumber` option, then the output files will have a suffix of `_split` followed by a split number (1, 2, 3 ...).

**-autoStraighten <max deviation> <min length>**

This specifies the auto-straightening algorithm should be run. It looks for sections of road, at least the minimum length long, where the road deviates laterally no more than the listed amount, each in meters. This is done before any other smoothing.

This may have mixed results. If a minimum deviation is too large, you'll potentially lose details of the route. 2 meters is a decent number. 4 meters is relatively aggressive.

This is in particular designed for routes generated from bike computer data, where rider position will move back and forth across the road, even for straight sections of road. However, if the road doesn't have long straight sections, then this will likely only decrease accuracy.

This option should likely be used in conjunction with some smoothing, for example by additionally providing the `-auto` option.

**-autoStraightenDeviation <meters>**

This is an alternative approach to specifying the maximum deviation for auto-straightening. See `-autoStraighten` for details. The default is 0 (do not do auto-straightening). Good options are 1 to 4 meters, from less aggressive to more aggressive.

**-autoStraightenLength <meters>**

The minimum length of a road section to be auto-straightened, in meters. This is an alternative to providing a second argument to an `-autoStraighten` option. It can be used in conjunction with `-autoStraightenDeviation` to request auto-straightening.

**-circle <meters> ... (1 or more)**

Provide segments to be fit with circles, alternating start distance from start of GPX, and finish distance from start of GPX. This is an alternative to providing values via `-circleStart` and/or `-circleEnd`. IF both methods are used, these will be processed first. So for example, `-circle 400 500 100 200` will fit circles for points between 400 and 500 meters from the start, then between 100 and 200 meters from the start. Not the further points are listed first so the fit of a circle to these points does not affect the distance to the nearer points. If an odd number of values are specified, the end point of the final point will be the end of the route.

**-circleEnd <meters> (1 or more)**

Set one or more distances for circle fitting.

See `-circleStart`

**-circleStart <meters> (1 or more)**

Set one or more start distances for circle fitting.

This is an option which should be used with care. For the special case where a route includes circular sections, for example laps of the San Francisco Polo Fields, you can move points to a circle which passes thru the end-points of an interval, and a point approximately mid-way between the end-points. This should be followed up with a bit of smoothing over the interval to clean up the transitions to and from the circular arc. Be careful in defining the endpoints so the direction doesn't change suddenly. For example, you don't want the circular arc to overshoot, requiring the direction to correct in the opposite direction. Consider only fitting the circular arc to a center portion of the curve.

The circular arc is fit between points starting at `-circleStart` and ending at `-circleEnd` from the start. If the `-isLoop` option is used, the start may come after the end in which case it wraps around. The circular arc is generated after point interpolation but before smoothing.

If you want to generate multiple circular arcs (for example, the Polo fields have circular arcs at each end, separated by straight segments) then specify the same number of values for `-circleStart` and `-circleEnd`, for example two values each for an oval like the Polo Fields.

If multiple values are listed, then multiple sections will be replaced. Distances will be updated after each circular segment replacement. So if you want to do multiple circular replacements, list them from further to closer relative to the start to avoid the distances for later fits being affected by earlier fits.

**-circuitFromPoint <meters> <circuits> [<repeats>] [<meters> ..]**

This is an alias for `-circuitFromPosition`.

**-circuitFromPosition <meters> <circuits> [<repeats>] [<meters> ..]**

This option allows you to specify a circuit starting at a certain distance from the start of the track. The code will go to a point that distance from the start. It will then search forward to where the course returns to the specified point and define that as a circuit.

If the number of circuits is greater than 1, it will repeat these points to create multiple circuits. If the number is zero, it will delete the points. If the number is one, then nothing will change.

An optional number of repeats may be specified if the circuit returns to this point more than once, in which case it must return to the original point the specified number of times.

**-circuitToPoint <meters> <circuits> [<repeats>] [<meters> ..]**

This is an alias for `-circuitToPosition`.

**-circuitToPosition <meters> <circuits> [<repeats>] [<meters> ..]**



This is like `circuitFromPosition`, except the distance specified is from the end of the course rather than the beginning, and the circuit ends at the specified point rather than starts there. So for example specifying a distance of 0 would search for a circuit which ends at the end of the track.

**-closed**

**-closedLoop**

**-closeLoop**

Equivalent to `-copyPoint -loop`. Negation negates `-copyPoint` and negates `-loop` unless it has already been set.

**-copyPoint**

For `-loop` courses, copy the first point to the end of the list of points, so the circuit is explicitly closed. If this is used without `-loop` then the result will be the same but smoothing will not extend across the S/F line. You probably want `-closeLoop` instead.

**-copyright <string>**

If you want to specify a copyright field in the GPX, list it here. The default will preserve the copyright of the source GPX.

**-cornerCrop <meters>**

If there is a sharp corner, defined as an angle in the range between `-minCornerCropDegs` and `-maxCornerCropDegs`, then points are placed before and after the corner and the corner point itself is deleted. The cropped corner is then rounded with a circular arc. Corner cropping allows for a reduction in smoothing and avoids `-minAngle` causing corners to bow outward. This is only done if the straight segments before and after the corner have sufficient length for the interpolated points to be placed.

**-cornerCropArcFit**

Whether to do arc fits when corner cropping. This is the default, so to turn it on use `-nocornerCropArcFit`. Without arc fitting, the cropped corner will be angular.

**-cornerCropEnd <meters>**

Where to stop corner cropping

**-cornerCropStart <meters>**

Where to begin corner cropping

**-cornerEffect <number>**

This adjusts the corner effect on gradient smoothing, which maintains the original gradient in corners. The default is 1. A nonpositive number turns off the experimental algorithm. A number between 0 and 1 makes it weaker. A number greater than 1 makes it stronger. In some cases using a value of 2 may help pin down a corner leading into or coming out of a steep climb, for example. However, larger numbers may lead to irregularities due to the rapid modification of smoothing.

This is not used in altitude or position smoothing.

**-crop <meters>**

This is a short-cut for the `-cropMax` option.

**-cropCorners <meters>**

This is an alias for `-cornerCrop`.

**-cropMax <meters>**

A point will be interpolated to this distance, if needed, and all points following will be discarded. Distances are calculated after smoothing but before `-append` or `-prepend`. It can

be useful to design a course beyond the desired length, smooth it, and then crop it, since smoothing can have anomalous effects near the boundaries of the data. This is especially true for a route ending on a section of road previously encountered in the route.

Since smoothing is affected by points ahead of and behind the given point, it is good to extend the points a few smoothing lengths, then crop it back to the desired endpoints.

Negative values are referenced to the end of the route. So for example, `-cropMax -140` would set a value 140 meters from the end of the route.

**-cropMin <meters>**

Points prior to this will be stripped, and if needed, a point will be interpolated to this position. This shifts the start position of the GPX route. See `-cropMax` for more discussion.

Negative values are referenced to the end of the route. So for example, `-cropMin -10000` would set a value 10 km from the end of the route.

**-cropStart <meters>**

Alias for `-cropMin`

**-cropStop <meters>**

Alias for `-cropMax`

**-crossingAngle <degrees>**

If set, then this is the minimum degrees by which segments need to intercept to be treated as crossings. If intercepting by less than this absolute angle, they are treated as separate. The default is 11.25. This can be increased since the default on twisty out-and-back sections might get marked as crossings otherwise.

**-crossingHeight <meters>**

If `-fixCrossings` is invoked, then the code will attempt to identify crossings and adjust the altitude at them. Typically crossings will be either level (same altitude each direction) or at some minimum height. That minimum height defaults to 2 meters, but can be adjusted with this parameter. If the GPX file has an altitude difference between zero and this number, it will force whichever is closer, and transition the altitude to either side, maintaining the same mean.

Altitude adjustment is handled by checking if the crossing elevations are close. If they are close, then they are averaged to make them equal. If they are not sufficiently close, they are stretched apart if they are not close enough for a reasonable overpass.

**-crossingTransition <meters>**

If `-fixCrossings` is invoked, then the altitude at crossings will be flattened over a length `rCrossings`. This flattening will be transitioned back to the original altitude over a transition length which defaults to 3 times `rCrossing`. However, this may be too short, resulting in excessive gradients. This option allows explicitly setting this transition length, specified in meters

**-csv**

Specifies the output will be CSV rather than GPX. This is ignored if an output file is specified with either a "csv" or "gpx" suffix (case insensitive), in which case the suffix is used to determine the format. Other suffixes are ignored.

**-curvatureSmoothing**

This controls how much curvature is smoothed before being applied to corner identification. The default is 10 meters. Increasing the number puts the emphasis on identifying larger-angle corners, and filtering out S-curves, for example.

**-curvatureThreshold**

This is the threshold of smoothed corners for the marking of corners, for example via the

`-addCornerWaypoints` option. Units are inverse meters. The reciprocal of this number is the maximum effective centerline turning radius of the corners to be marked. The default is 0.05, avoiding gradual corners. Note if `-minRadius` or `-smooth` are set too high, no corners will be identified unless this value is decreased.

**`-deleteRange <meters> <meters> ...>`**

This allows the specification of position pairs marking ranges over which points should be deleted. If two points are provided, then the route is deleted starting at the first position and ending at the second position. Additional pairs may be provided to provide additional delete ranges. This deleting, unlike the deleting specified with `-deleteMin` and/or `-deleteMax`, is done before point interpolation, but after file joining. This is because it is likely that smoothing will need to be done after the delete. Additional clean-up with a GPX editor like GPXMagic may be useful.

This would be useful, for example, if a GPX file has an out-and-back portion that you want to remove from the route.

If multiple segments are provided the distances are not adjusted between deleting operations. So for example, if `-deleteMax 1000 2000 3000 4000` were specified, the section from 1000 meters to 2000 meters would be first deleted. This would reduce the distance to the point which was originally at 3000 meters. However, the point at 3000 meters is determined using the distances calculating before any deleting was done. Delete segments should not overlap. Distances should be specified with extreme care, ideally to provide a small gap between the end of what precedes the start of the delete, and what follows the end of the delete, to allow a smooth transition.

If there is only a single position provided, or if there are an odd number of positions, then the only or odd position will be considered a minimum position for deleting. So, for example, `-deleteRange 10000` would delete the route starting at 10 km, where positions are determined on the unprocessed GPX data.

If the second point of a range is less than the starting point of the range, then the range wraps around, so points at the end of the route (after the start point) and at the beginning of the route (before the finish point) may be deleted.

Deleting ranges doesn't create gaps in the route, of course. It deletes the control points within a given distance range. So the points at either end of the range will be connected. So for example, suppose I have a straight-line course with points at 0 meters, 500 meters, and 1000 meters. I then delete in the range 250 meters to 750 meters. I will now have points at 0, 250 meters, 750 meters, and 1000 meters. The points at 250 meters and 750 meters have been interpolated to define the deletion range, and the point at 500 meters has been deleted.

**`-description <string>`**

List a description for the GPX metadata. Since the description will likely contain spaces, remember to enclose the string in "quotes", or however else your command-line shell delimits spaces.

**`-extend <meters>`**

This is a simultaneous specification of both `-prepend` and `-append`.

**`-extendBack <meters>`**

This option creates a turn-around loop at the end of a point-to-point route and then truncates it so the extension in length is the specified number of meters. So, for example, `-extendBack 200` will result in a route 200 meters longer, where the additional length is provided by a turn-around loop, then added distance as needed backtracking along the main route.

This option was added for RGT, which used the final 140 meters of a point-to-point route as a zone for riders to group after the finish.

This disables `-loop` or `-lap`: it's meaningless with a lap course

**`-finishCircuitDistance <meters>`**

Often races end with finishing circuits after a lead-in. To accomodate this, you can start with a GPX route which includes at least one lap of the finishing circuit, and add finishing loops. This option allows specifying the start position where a lap of the finish circuit begins. The circuit is assumed to extend to the end of the GPX file. The code will add a number of copies of these points specified by the `-finishCircuits` option.

So if the course goes from A to B, then completes 3 circuits of the loop B-C-D, then find the distance in meters to point B, then specify that distance as `-finishCircuitDistance`, and specify `-finishCircuits 2`. This will add two copies of the loop B-C-D to the end of the data.

If the desired finish of the GPX is not at the same point as the end of the loop, then you'll need to specify a `-cropMax` value to remove some of the final circuit.

An alternative to this option is the newer `-circuitToPosition` which if specified with a position of 0 will generate finishing circuits without having to identify the position where the circuits begin. So for example `-circuitToPosition 0 3` would generate three finishing circuits automatically.

If `-finishCircuits` is specified but not `-finishCircuitDistance`, then the result is equivalent to `-circuitToPosition` to position 0 (counting from end) with one repeat (meaning the circuit starts the penultimate time it reaches the finish point).

#### **`-finishCircuits <count>`**

A finishing circuit starting at the position specified by the `-finishCircuitDistance` option, in meters, will be appended to the initial data this number of times. A value of 1 here implies adding one copy, which will result in two laps of the circuit.

If `-finishCircuits` is specified but not `-finishCircuitDistance`, then the result is equivalent to `-circuitToPosition` to position 0 (counting from end) with one repeat (meaning the circuit starts the penultimate time it reaches the finish point).

#### **`-finishCircuitStart <meters>`**

Synonym for `-finishCircuitDistance`.

#### **`-fixCrossings`**

If a route contains crossings, for example a true figure 8, then it will flatten the road on either side of the crossing, and create a transition from the flattened profile back to the unaltered profile. The side of the flattening is determined by the `-rCrossing` option. An issue with RGT is that if a sloped road crosses a flat road, for example, on the upward side the sloped road will appear to rise above the surface, while in reality the pavement would be leveled out to allow for the crossing. This creates that leveling.

#### **`-fixSteps`**

Strava route editor is prone to produce points where proximate points have identical altitude, likely because of missing data. This option applies removes the step by applying a uniform gradient to merge the flat segment with the steeper of the two adjacent sides. Of course this is done early in the process, before any interpolation or smoothing.

If there are points with the same altitude you wish to protect from this process, then one approach is slightly change the altitude of the points, for example with the 1 cm "nudge" option in GPXMagic, to avoid equivalent altitudes.

This option is not invoked by `-auto`, so if you are using Strava Route Editor to generate GPX files, you will probably want to specify this option in addition to a possible `-auto`.

#### **`-flatten <start-meters> <altitude-meters> <end-meters> <altitude-meters> <transition-meters> <bow-meters> ...`**

This option allows the route to be flattened over a specified coordinate range, with a specified transition length. This has been used where there are overpasses in a route, where it is important that the altitude difference between the upper and lower roadway be maintained, even with altitude smoothing.

You specify a starting distance, then the altitude at that point, then a finish distance, then (optionally) an altitude at the second point (default is the same altitude as the first point), then optionally a transition length (default is to calculate a reasonable one).

The transition length is used to describe a distance over which a cosine weighting term is used to transition between the fixed altitude, and the prior altitude for those points. This should be made long enough to avoid excessive deviations in gradient.

Bow describes how far up (positive) or down (negative) the center of the segment is raised relative to the straight line connecting the endpoints. This may be useful for suspension bridges, for example.

Multiple sets of six numbers can be provided, in which case altitude flattening is done over each of the specified segments.

Distances are calculated *before* most route processing, after possible route reversal. So if you are using `-reverse`, specify the distance values from the start of the route after reversal. It is done before possibly adding circuits, so the adjusted altitudes will be available on all circuits.

If you want a uniform gradient but do not want to specify the endpoint altitudes, consider `-uniformGradient` instead.

#### **-gAutoSmooth**

Synonym for `-autoSmoothG`

#### **-gSmooth**

Synonym for `-smoothG`

#### **-gSigma**

Synonym for `-smoothG`

#### **-gradientPower**

For gradient signs, how much of a power to apply to gradient in determining where signs go. If 0, then all that matters is altitude: put the signs between peaks and valleys. If 1, then a climb which is double the altitude but half the gradient scores the same. The higher this number, the more likely a gradient sign is to go on a short steep pitch versus a longer, more gradual climb containing the short steep pitch.

#### **-gradientThreshold**

This determines the threshold at which a gradient sign gets put in. The units are meters: a 10% climb needs to gain this much altitude to get a gradient sign. How much altitude steeper or less steep climbs need to be depends on `gradientPower`

#### **-interpolate <meters>**

Synonym for `-spacing`

#### **-join <filenames>**

Add the points from the first track found in these files and append them to the selected track in the original file. The files must reasonably match up end-to-end. To specify alternate tracks than the first, append the selected track after a colon (":"). For example, `-join file.gpx:2` would select the second track in file `file.gpx`.

#### **-keywords <meters>**

Add keywords to the GPX output. Multiple keywords can be separated with commas. If there are any spaces, make sure to enclose the string in quotes, or however your command-line shell specified strings should be delimited.

So for example, the following are allowed:

`-keywords test`

`-keywords test1,test2,test3`

---

`-keywords "test1, test2, test3"`

A "processGPX" keyword is automatically added.

**`-laneShift <meters>`**

This shifts the points of a road to the right (for a positive value) or the left (for a negative value). This is used for out-and-back sections, to provide separation between the outward and return legs of the road, which are otherwise described with the same coordinates. In RGT Cycling, a 4 meter shift will cause the resulting inward and outward roads to abut, assuring that even if cyclists use the full road width, they will not visually collide. A larger value may result in a grass island between the two directions. A smaller value may result in the roadways overlapping.

Back in the day a 4 meter laneshift was sufficient. But then RGT added road-side assets to magic roads, so a 4-lane shift with abutting roads will result in riding through tents, fences, etc, associated with the opposite direction. A 6 meter lane-shift is the minimum recommended, or even more to avoid issues with the large tents. Hopefully RGT will fix this problem.

**`-laneShiftEnd <meters>`**

This is an alias for `-shiftEnd`.

**`-laneShiftStart <meters>`**

This is an alias for `-shiftStart`.

**`-laneShiftTransition <meters>`**

This is an alias for `-shiftTransition`.

**`-lap`**

Synonym for `-loop`

**`-loop`**

The course is considered a loop course, or a circuit, and so smoothing and other operations can take place between the beginning and end of the loop.

**`-loopLeft`**

Specify that turnaround loops should loop to the left (UK, for example). The default is to base it on the `-laneShift` (left for a negative lane shift, else positive).

**`-loopRight`**

Specify that turnaround loops should loop to the right (US, for example). The default is to base it on the `-laneShift` (left for a negative lane shift, else positive).

**`-maxCornerCropAngles <degrees> =item -maxCornerCropDegs <degrees>`**

The maximum angle at which corner cropping should be done, for example 120 degrees.

**`-maxSlope <%>`**

RGT supported a "maximum slope" option for magic roads. This option allows that to be changed from a default. It is specified in percent. If you specify a number < 1, then that will be assumed to not have been converted into percent, so the number will be multiplied by 100. With the closure of RGT it is unlikely this has any remaining utility.

**`-minCornerCropAngle <degrees>`**

**`-minCornerCropDegs <degrees>`**

The minimum angle at which corner cropping should be done, for example 60 degrees.

**`-minRadius <meters>`**

The code will calculate an effective radius of turns, and if this is less than this value, it will shift the road to increase the radius. This is done prior to `-laneShift`, which may thus result in

tighter turns. So if there is a tight switchback, for example, then applying this option will tend to shift the road outward, increasing the turn radius. There is a transition for this lane shift, so for tight S-turns, alternating left and right, or in very tight switchbacks, where the road snakes up a hill, results may be unsatisfactory. It works best with an isolated corner.

It generally helps to add `-cornerCrop` with a number at least equal to the minimum radius to avoid corners extending outward.

**-minRadiusEnd <meters>**

If specified, where to stop applying minimum radius (`-minRadius`). If both `-minRadiusStart` and `-minRadiusEnd` are specified, and if the end is before the beginning, then the region between the points is excluded, and it is applied to the region after the start and the region before the end. To apply multiple radii in different regions of the course, this can be used for one region, but then the code should be re-run on the resulting GPX file to apply a subsequent region.

**-minRadiusStart <meters>**

If specified, where to start applying minimum radius (`-minRadius`). If both `-minRadiusStart` and `-minRadiusEnd` are specified, and if the end is before the beginning, then the region between the points is excluded, and it is applied to the region after the start and the region before the end. To apply multiple radii in different regions of the course, this can be used for one region, but then the code should be re-run on the resulting GPX file to apply a subsequent region.

**-name <string>**

Specify the name of the GPX route. The default is the name listed in the source GPX. This is synonymous with `-title`.

**-noSave**

The `-nosave` option suppressed generation of a GPX output file. This may be useful for debugging or for checking the distance of a file, which is reported to standard error, and checking it for zig-zags and loops.. This option has a double-negation which is `-nonoSave`.

**-o <filename>**

Alias for `-out <filename>`

**-out <filename>**

Instead of the default filename, which is the input file with `_processed.gpx`, use this filename instead. Make sure to specify the `.gpx` suffix if that is what is wanted. A "-" implies standard output.

**-outAndBack**

This is a short-cut for creating an out-and-back, with a loop at the end of the magic road, then a return. It defaults to shifting the road to the right. If you want the left (UK, for example) then specify the `-laneShift` option as well. It should obviously not be combined with `-lap` unless you add `-rLap`.

If you want a different lane shift than the default 4 meters, or a different loop radius than the default 8 meters, you can specify these with `-laneShift` and/or `-rTurnaround`. You probably also want to add some smoothing, so `-auto` is a good option.

If you use this option, then the starting route should be just the "out" section, not the "back". The code will generate the turn-around loop and the return road, which will be shifted.

To avoid issues with sharp corners, `-cornerCrop` and `-minRadius` are set by default. You can tune these parameters if corners aren't what you want.

This option is generally not needed with BikeTerra which automatically forms turn-around loops at each end of a point-to-point course.

For routes with sharp corners, for example tight switchbacks, this can give highly anomalous

results due to the lane shift. You can try adding a lot of smoothing, for example `-smooth 20`.

#### **-outAndBackLap**

This is the same as `-outAndBack` (read that section as well), except also adds a turn-around to create a lappable course. If you want the turnaround and lap loops to be different than the default radius, then you can specify these separately with `-rTurnaround` and/or `-rLap`.

This option is not generally needed with BikeTerra which automatically forms turn-around loops at each end of a point-to-point course.

#### **-outAndBackLoop**

An alias for `-outAndBackLap`

#### **-prepend <meters>**

See the `-append` option for details, except instead of adding roadway to the end of a point-to-point course, this adds it to the beginning of the course. RGT in general puts the start line 60 meters after the start of a GPX file, so this can provide for that distance, although a better solution is to design in the 60 meter buffer from the start, so it conforms to the actual roadway.

A negative number will crop the course at the beginning, so is an alternative to `-cropMin`.

#### **-prune**

This option says that colinear points (in all three dimensions) should be removed, reducing the size of the output file. There's no downside to this, unless the file is being prepared for subsequent modification with another tool, such as GPX Magic.

#### **-pruned <meters>**

Given three points in sequence, A, B, and C, if the separation between B and the segment connecting A and C is at least this distance, the point B is not pruned. The `-prune` option must still be invoked to get pruning. There is a default value.

#### **-prunedg <value>**

Given three points in sequence, A, B, and C, if the difference in gradient from A to B, and B to C, exceeds this value (not specified as a percent, but as a raw value). The default is 0.0005.

#### **-pruneDistance <meters>**

An alias for `-pruned`.

#### **-pruneGradient <value>**

An alias for `-prunedg`

#### **-pruneSine <value>**

An alias for `-pruneX`

#### **-pruneX <value>**

Given three points in sequence, A, B, and C, if the sine of the angle A-B-C exceeds this value, the point B will not be removed. There is a default value.

#### **-quiet**

If specified, suppress all noncritical messages (to the standard error stream). Only warnings and error messages will go to the standard error stream. This will suppress things like the number of points, the altitude smoothness score, and the course distance.

#### **-rCrossings <meters>**

If `-fixCrossings` is invoked, then how far to each side of the crossing the road should be leveled. A transition of three times this length will be used, over which the altitude will be restored to the unaltered value. The default is 6 meters, which works fairly well with the RGT



road width of 8 meters, although if the intersection is at a particularly acute value, or if the roads will be wider than the RGT standard of 8 meters, a larger value may be better. If the value is too large, the transitions too and from the flat section may be too steep, or the influence of crossings could overlap, which the algorithm does not handle well.

**-repeat <count>**

Number of copies of the route to append to the end of the course. This should naturally be applied to a route which is a lappable course. Normally lappable courses in RGT are modeled with only a single lap, but there are several reasons one might want to artificially create multiple laps by repeating the lap. One might be if one is going to add a lead-in or finishing leg to the course, for example with the `-join` option, which would be applied after this command is executed. Another might be if you wanted timed segments only on selective laps, or overlapping the boundary between laps. But in most cases multi-lap races are better modeled by specifying the number of laps at event creation and using a magic road which covers only one of the laps.

**-repeatedLaps <count>**

This is equivalent to `-repeat` except with a number one less. For example, `-repeatedLaps 10` and `-repeat 9` are the same. Each results in 10 copies of the original GPX file.

**-reverse**

This reverses a course immediately after loading the file. So all subsequent operations will be done on the reversed course rather than the original course. So for example, if I have an initial course which goes from point A to point B, and I specify a turnaround with `-rTurnaround 5`, instead of a route from A to B and back, it will instead be from B to A and back. If you want to reverse the final product, then do a second run with just the `-reverse` option after an initial run on other options.

**-rLap <meters>**

For an out-and-back course, create a second loop at the finish, reconnecting to the start, to create a circuit. This allows the out-and-back to be repeated an arbitrary number of laps. This only works if `-rTurnaround` is also positive. See also `-rUTurn`.

If you want to go from a single point-to-point course to get an out-and-back loop course, consider the `-outAndBackLap` option, which specifies a range of options including this one.

**-rTurnaround <meters>**

Create an out-and-back course, with a turn of this radius generated at the turn-around point. This may be done in conjunction with `-laneShift` to have the return road shifted from the outward road. If the distance is less than the `laneShift` value, then the loop will have 3 parts: for example a right, then a left, then another right to turn 180 degrees.

The default behavior is for the loop to be to the left if `laneShift` is negative (UK, for example), otherwise positive (US, for example). The direction can alternately be specified with `-loopLeft` or `-loopRight`.

The `-outAndBack` option is an alternative, specifying other smoothing options in addition to a lane shift and this option to create an out-and-back course more simply.

This option should not be used with BikeTerra which automatically forms turn-around loops at each end of a point-to-point course.

**-rUTurn <meters>**

If any 180-degree turns are identified in the course, loops are added with this radius.

This is done late in the process, in particular after lane shifting, so the U-turns can properly connect the land-shifted roads (with the `-laneShift` option). It is done before `minRadius`, however.

**-saveCrossingsCSV**

If crossing processing is done with `-fixCrossings`, then this option specifies a CSV file should be saved with a similar filename with coordinates of detected crossings. It's useful for debugging, since the crossing detection algorithm is imperfect.

**-saveSimplifiedCourse**

This is an option for debugging. `-fix_crossings`, for example, generates a simplified course, and if find in a complex course that certain crossings aren't identified, then this may be useful for debugging why.

**-segment <start>,<end>,<name>; ...**

Define one or more non-overlapping named segments for the route. These are defined immediately before the `-shiftSF` option is applied.

Segments are specified with groups of 3 options separated by comma: a first coordinate specifying the starting distance in meters or "start", a second ccoordinate specifying the ending distance in meters or "end", and a name. The starting coordinate instead be "start" to begin at the course start, and the ending coordinate can instead be "end" to end at the course end. It is not recommended to use "start" or "end" with a loop course, since for loops, the course will not start or finish at the GPX end.

RGT requires that segments start a sufficient distance from the beginning of the route, and end a sufficient distance from the end of the route, and also be separated by a sufficient distance. However, this code only checks that they not overlap. There is no way to define segments which overlap the S/F of a lapped course.

Segments already defined in the GPX file, for example from GPX Magic or previous runs of this code, are honored, and so new segments may not overlap previously defined segments. Points are interpolated to get an essentially exact match to the specified coordinates.

Consider the `-stripSegments` option to merge segments from the input file if the input file has multiple segments and you wish to replace them.

If you want multiple named segments, then split them with semicolons (";"), so for example  
`-segments "1000,2000,segment 1;3000,4000,segment 2"`

In many cases, the `-autoSegments` option is a better option, since this will automatically find segments by searching for climbs within a track.

**-selectiveGSmooth <meters> ...**

This is an alias for `-selectiveSmoothG`, which provides for selective gradient smoothing.

**-selectiveLaneShift <meters> ...**

This allows for a lane shift which is varied throughout the course. The first number is the lane shift for the first part of the course. The next number is a position where the lane shift changes to the next smoothing number. A third number is the lane shift beyond the second number. This can continue. You alternate lane shift numbers and positions marking the transition between these shifted numbers. So there should be an odd number of numbers following. If there's an even number, a last number of zero is assumed (no shift after the listed distance).

For example, suppose I want to use 7 meter shift up to 2 km, then change to 2 meter shift from 2 km to 3 km, then back to 7 meters past 3 km.

`-selectiveLaneShift 7 2000 2 3000 7`

the lane shift will be smoothly varied between values at the boundaries using a cosine transition.

If this is a lapped course, and there are multiple lane shift values, then there will be an implicit transition at the S/F of the lap unless the last number matches the first.

If you specify lane shift as well as selective lane shift, then both are applied simultaneously (distances refer to positions before lane shifting is applied).

**-selectiveSmooth <meters> ...**

This allows for smoothing to be varied throughout the course. The first number is the

smoothing distance for the first part of the course. The next number is a position where the smoothing changes to the next smoothing number. A third number is the smoothing beyond the second number. This can continue. You alternate smoothing numbers and positions marking the transition between these smoothing numbers. So there should be an odd number of numbers following. If there's an even number, a last number of zero is assumed (no smoothing after the listed distance).

For example, suppose I want to use 7 meter smoothing up to 2 km, then change to 20 meter smoothing from 2 km to 3 km, then back to 7 meters past 3 km.

```
-selectiveSmooth 7 2000 20 3000 7
```

#### **-selectiveSmoothG <meters> ...**

This is similar to `-selectiveSmooth` except it is applied only to gradient smoothing.

#### **-selectiveSmoothZ <meters> ...**

This is similar to `-selectiveSmooth` except it is applied to altitude only. This is particularly useful with Strava Route Editor output, where the quality of the altitude data often varies.

For example, suppose I want to use 15 meter smoothing everywhere except between km 4 and km 5, where I will use 50 meter smoothing, knowing perhaps from bike computer data that the gradient in that interval is much more uniform than is reflected in the Strava Route Editor data:

```
-selectiveSmoothZ 15 4000 50 5000 15
```

#### **-selectiveZSmooth <meters> ...**

an alias for `-selectiveSmoothZ`

#### **-selectiveSpacing <meters> ...**

This applies selective point spacing on the course before other processing. It is done in addition to the `-spacing` and `-autoSpacing` specifications. If the specified value is larger than the spacing implied from those options, then there is little effect, since those will be applied first. If the spacings in this option are smaller, then they will be applied after those values, possibly further reducing the point spacing.

Values are specified in groups of three, with the first being a spacing in meters, the second being a start of the interval for this spacing, and the third being the end of the interval for this spacing. Additional sets of three points can then be specified. On loop courses only the end point can be before the start point, in which case the interval will wrap around the S/F point.

The following is an example of applying a 1 meter spacing to the region from 1000 meters to 1100 meters, then 2 meter spacing from 1100 to 1200 meters:

```
-selectiveSmooth 1 1000 1100 2 1100 1200
```

On many commands involving intervals, if the stop point is before the start point, that implies the interval is after the start point and before the end point but not between. On point-to-point courses that makes no sense here, so these intervals are rejected on point-to-point courses. Of course on loop courses (`-loop`, `-lap`, or `-autoLoop`) wrapping around still makes perfect sense, so this is still the interpretation.

#### **-shiftEnd <meters>**

The position at which lane shifting should end. This is useful in case you have an isolated out-and-back section and want to be able to shift lanes over only a portion of a course, or for example for an out-and-back where there will be little risk of head-on collisions sufficiently far from the turn-around. A transition zone between shifting and non-shifting is created. The distance is calculated prior to lane shifting, not self-consistently (lane shifting will subtly change distances), so to determine precise distance, run once without lane shifting, get the distance, then do lane shifting with that precise distance.

A reason for limiting lane shifting is that lane shifting creates a deviation in the route from the real-world coordinates, unless the road is sufficiently wide, and also because it can cause a

decrease in radius of tight turns to too small value. Lane shifting should thus be combined with a sufficient degree of position smoothing to avoid tight corners, or else restrict the lane shifting from a portion of the course with tight corners.

If `shiftStart < shiftEnd`, then the shift occurs between `shiftStart` and `shiftEnd`.

if `shiftStart > shiftEnd`, then the shift occurs up to `shiftEnd`, then begins again at `shiftStart`. This is useful for a "lollipop" course, where you go out on a road, then do a loop, then return along the original road. So put `shiftEnd` at the end of the outward leg, then `shiftStart` at the beginning of the return leg, and leave the loop part unshifted. Place the transitions slightly past where the directions diverge, so the transition regions do not cause the lanes to come together at the beginning of the return leg (end of the outward leg).

The `-laneShift` option still needs to be specified.

#### **-shiftStart <meters>**

The position on the course where lane shifting starts. See `-shiftEnd` for details, except this marks the start of the lane shift zone.

The `-laneShift` option still needs to be specified.

#### **-shiftTransition <meters>**

The distance to each side of lane shift transition points to taper the lane shift. Defaults are normally fine for this, so this option may never need to be specified. For `-shiftStart` and `-shiftEnd`, which are applied to `-laneShift`, the transition is calculated based on the lane shift magnitude. For `-selectedLaneShiftC`, the default is 20 meters.

#### **-shiftSF <meters>**

For `-loop` courses, the amount to shift the start/finish forwards (positive) or backwards (negative).

The default is `-shiftSF 0`, which retains the original GPX start.

This was needed with RGT courses to accommodate the 60 meter start zone. However, BikeTerra puts the S/F at the starting point for the GPX track, so no shift is needed.

#### **-shiftX <meters>**

This is an alias for `-xShift`.

#### **-shiftY <meters>**

This is an alias for `-yShift`.

#### **-shiftZ <meters>**

This is an alias for `-zShift`.

#### **-shiftZEnd <meters>**

This is an alias for `-zShiftStart`.

#### **-shiftZStart <meters>**

This is an alias for `-zShiftEnd`.

#### **-sigma <meters>**

Synonym for `-smooth`

#### **-sigmag <meters>**

Synonym for `-smoothG`

#### **-sigmaz <meters>**

Synonym for `-smoothZ`

#### **-simplify**

---

Synonym for **-simplifyPoints**.

**-simplifyAltitude <meters>**

Synonym for **-simplifyZ**

**-simplifyD <meters>**

The maximum deviation of a point from a straight line for simplification. So simplifying a segment would result in a point deviating further from this in x, y coordinates, then the point will be retained.

**-simplifyDistance <meters>**

Synonym for **-simplifyD**

**-simplifyPoints**

This uses a Ramera-Douglas-Peucker algorithm to recursively simplify the points. It results in more aggressive point removal than just -prune. If point count is at a premium, this is a good option in addition to pruning.

It is suitable for BikeTerra which does spline interpolation of points, in contrast to RGT, which linearly interpolated points and thus required more points to create smooth corners.

**-simplifyZ <meters>**

The maximum deviation of a point's altitude from a uniform gradient for simplification. So simplifying a segment would result in a point at least this amount higher or lower than for a course without the point present, then the point will be retained.

**-smooth <meters>**

Provide a Gaussian sigma value for smoothing of position and altitude. The result will sharp corners will be rounded to corners with approximately this radius, and grade fluctuations over less distance than this will be lost. Typically altitude needs more smoothing than position, so additional altitude smoothing is required.

**-smoothing <meters>**

Synonym for **-smooth**

**-smoothAngle <degrees>**

For **-autoSpacing**, determines how dense to place points such that the maximum angle between points is no more than approximately this angle.

**-smoothEnd <meters>**

See **-smoothStart**. The default is to continue smoothing to the end of the GPX data.

**-smoothG <meters>**

Additional smoothing to be applied to gradient, after smoothing applied to position and/or altitude. Gradient smoothing is very similar to altitude smoothing it smooths the gradient, then calculates altitude from the new gradient numbers, adjusting slightly to restore net altitude change for the route.

There's a few differences from altitude smoothing. One is that altitude smoothing, on point-to-point courses, tends to flatten out the beginning and end of the route, while gradient smoothing tends to maintain the gradient at the beginning and end. A second difference is that gradient smoothing automatically reduces smoothing in corners with the philosophy that on real roads the gradient in corners may differ from the gradient on the straight road either before or after, and preserving that detail is more realistic. Altitude smoothing smooths corners and straight sections the same.

The corner effect term can be adjusted with the **-cornerEffect** option.

Altitude smoothing is better at preserving altitudes, while gradient smoothing is more focused on gradients, although since the two are related by a derivative, the difference is extremely

subtle. This is a more experimental algorithm than altitude smoothing, due in part to the error correction at the end, but more for the consideration of cornering radius.

**-smoothStart <meters>**

If specified, smoothing will be phased in starting at this position in the course. Note 0 has an effect, as it will phase in smoothing starting at 0, as opposed to the default, which is to apply the same smoothing to all points. This applies to all smoothing, so consider running processGPX multiple times if you want different limits on different smoothing components. For example, you might want to exclude position smoothing from critical corners, but maintain altitude smoothing. If the start is after the finish, then the region between the two is not smoothed, rather than smoothed.

**-smoothZ <meters>**

Additional smoothing to be applied to altitude, on top of the smoothing applied with `-smooth`. So this number, if specified, will typically be greater than the `-smooth` number or there is little effect. Grade changes which occur over less than this distance will tend to be averaged out. So for example, if I am designing an urban course, and there is a turn onto a sharp climb, then less smoothing can be tolerated. On the other hand, if I am designing a course with a steady grade up winding switchbacks on a steep hillside, then more smoothing be needed. On roads along steep hillsides, altitude accuracy is more challenging than it is on roads which take the direct route up more gradual hillsides.

**-snap <option>**

With snapping, the code will search for sections of road which repeat, either in the forward or reverse directions, and "snap" one pass to the points of the other pass, guaranteeing that the two are perfectly aligned. So for example, if a route covers 1.5 laps of a course, something not presently supported by the RGT multi-lap option (which handles only complete laps), then snapping will make the final half-lap the same as the first half-lap, up to within close to the end of the route (since smoothing is affected by proximity to an end of the route). Similarly if a route has an out-and-back section, this will help make sure there's no altitude or position differences between the two.

**option 1:** later passes are "snapped" to earlier passes.

**option 2:** earlier passes are "snapped" to later passes.

Sometimes one or the other will work better in a particular case, depending on whether an earlier or latter pass over a section of road has better definition.

If there are existing segments, snapping will try to map segments from the old points to the new points, but it's more precise to define segments after snapping. If you define segments within the same run of processGPS, they will be done after possible snapping.

Snapping is done both before and after point interpolation. The reason is it's potentially very efficient with a smaller number of points, but sometimes more widely spaced points make it more challenging for the algorithm to determine that a section of road is being repeated. In the past, when snapping was done only once, it was a regular practice to run processGPX twice in an effort to more effectively snap points.

The snapping algorithm does not work at the very start or end of the GPX. So if your route ends on a repeated section and you want it to be snapped, then it's a good practice to extend the route further, then crop it back with `-cropMin` and/or `-cropMax`.

**-snapAltitude <meters>**

Normally for snapping to identify two sections of road as being a repetition, they need to be within 1 meter altitude, to avoid "snapping" on very tight switchbacks, for example, where roads may be close on a map but at different elevations. But in cases where map data have elevation errors, this may prevent legitimate snapping. This parameter allows for the altitude tolerance to be increased from the default, relatively tight, 1 meter limit. Note there's an additional tolerance for the separation, using a 30% gradient, which is not adjustable.

**-snapDistance <meters>**

The distance in meters a road segment can deviate from another and still be "snapped" (see the `-snap` option). This example can be important: if snapDistance is too small, instead of the repeated road being replaced in one piece, it may be fragmented. Look to the standard error for text describing which segments have been snapped. For example, the following from a criterium course with an out-and-back section:

"snapping reverse segment: iRange = 25 35 <=> jRange = 54 47"

This is a nice clean replacement

**-snapTransition <meters>**

This is an *experimental* feature whereby points within this distance of a snap transition on partially synchronized in altitude with a zero gradient, with a transition weighting depending on geometrical distance from the snap transition. This is designed to avoid having divergent roads having different slopes while still overlapping. It's experimental, however, and there is no guarantee it will help rather than hurt, because of the way smoothing affects points near snap transitions, and snapping is almost always combined with smoothing to avoid abrupt jumps in position.

**-snapZ <meters>**

This is a synonym for `-snapAltitude`

**-spacing <meters>**

As an early stage to processing, interpolate points on the route so that the spacing between points is no more than approximately this spacing. If `-smoothing` and/or `-smoothingz` are specified, then smoothing doesn't work over distances much smaller than this spacing.

`-autospace` is another option, in which case the code will selectively interpolate points near points where the direction is changing.

**-splice <filenames>**

Starting with the points of the first file, find compatible segments associated with the spliced files, in order, and replace those segments with the points from the spliced files. There is no transition so if the points do not fit perfectly some clean-up may be needed, for example smoothing or point editing.

This may be used, for example, to replace points on a particular climb within a longer route with points for the same climb from a second, superior source. For example, GPX data may be poor for the climb from a mapping program, but you may want to use points from a GPS bike computer instead for a climb. Just make sure that the new points align well with the original track.

If the route is a loop, and if the splice match wraps around, then the S/F will be chosen as the point on the splice which is closest to the original S/F point.

For point-to-point races, the splice must be contained within the original route. It cannot replace the first or last point, for example. It also cannot extend the route. For loops, the splice can fit anywhere in the loop, including crossing the original start/finish. The new start/finish will be the point in the spliced data closest to the original start/finish.

Splicing is done early, before any point interpolation, but after a possible `-autoLoop@` option is processed.

Splicing will only occur if the start of the splice is within a certain distance of an endpoint from the original file. This distance defaults to 250 meters, but can be set with the `-spliceDistance` option.

**-spliceDistance <meters>**

This allows specification of how close the spliced segment must be to a point in the original file to be considered a candidate for splicing. The default is 250 meters.

**-splineAngle <degrees>**

**-splineDeps <degrees>**

If splines are desired, specifying this will cause spline interpolation to be done for corners turning at least this much, but less than the `-splineMaxDeps` option. A typical value is 5 degrees. This angle is the angle between course points, not an angle of a total turn: the algorithm only looks at points ahead of and behind a given point.

**-splineEnd <meters>**

Where to end spline fitting, if any. The default is to end any spline fitting at the end of the course. This can "wrap around".

**-splineMaxAngle <degrees>****-splineMaxDeps <degrees>**

If a `-splineDeps` option is specified, specifying this limit the maximum angle corner for which spline interpolation will be applied. Splines are good for gradual, rounded corners but are not good for sharp corners, so an upper bound in the 60 degree range (which is the default) works generally well. Splines have the advantage of rounding corners without "blunting" them, but sometimes they create "S" shapes where they are not wanted. Make sure to check the results if using spline interpolation: strange results can occur if the corner is too sharp.

**-splineStart <meters>**

Where to start spline fitting, if any. The default is to start any spline fitting at the start of the course. This can be after `-splineFitEnd` and the region will wrap-around.

**-splitAt <meters> ...**

Specify one or more locations in the final route, measured in meters distance from the GPX start, at which the route should be split.

If you specify an output file and do not specify a specific split number with the `-splitNumber` option, then the output files will have a suffix of "\_split" followed by a split number (1, 2, 3 ...).

If you want the original route to be split into a number of equal pieces, then see the `-autoSplit` option instead.

**-splitNumber <1, 2, 3..>**

If there are multiple splits (`-splitAt` or `-autoSplits`), which splits to output. The default is to output all splits if a filename is specified for the output (using a `_split` with the split number as a suffix) or just the first split if it's to standard output ("-"). The default can be specified with a non-positive number, or just omitting this option.

**-startCircuitDistance <meters>**

Sometimes races start with multiple circuits of a loop, before leaving the circuit for a remainder of the course. This option allows you to repeat a beginning portion of the GPX file as a finishing circuit. To do this, you specify the distance to the end of the circuit, then use the `-startCircuits` option to specify how many copies of this circuit should be prepended to the route at the beginning.

If the start/end of the circuit lap is not the same place as the desired beginning of the route, then you'll need to specify a `-cropMin` value to remove some of the initial circuit

If `-startCircuits` is specified but not `-startCircuitDistance`, then the result is equivalent to `-circuitFromPosition` from position 0 with one repeat (meaning the circuit ends the first time it returns to the starting point).

**-startCircuitEnd <meters>**

Synonym for `-startCircuitDistance`.

**-startCircuits <count>**

The number of copies of the starting circuit (from distance 0 to the value in meters specified



with the `-startCircuitDistance` option) to be added to the beginning of the data. So a value 1 means adding one copy, which implies two laps of the circuit, including the one defined in the original file.

If `-startCircuits` is specified but not `-startCircuitDistance`, then the result is equivalent to `-circuitFromPosition` from position 0 with one repeat (meaning the circuit ends the first time it returns to the starting point).

**`-startTime` "<time string>"**

Specify a start line for an activity using clear notation, for example:

```
processGPX -startTime "15 Feb 2021 08:00"
```

would generate a time field beginning at that time, in the local time zone. This is useful for uploading a GPX route to "Relive", a website which generates animations of routes, and requires a time field. The time is generated using a heuristic formula which has rider speed depend on the road grade, calibrated for a strong rider.

**`-straight` <meters> (1 or more)**

Provide segments to be fit with straights, alternating start distance from start of GPX, and finish distance from start of GPX. This is an alternative to providing values via `-straightStart` and/or `-straightEnd`. If both methods are used, these will be processed first. So for example, `-straight 400 500 100 200` will fit straights for points between 400 and 500 meters from the start, then between 100 and 200 meters from the start. Note the further points are listed first so the fit of a straight to these points does not affect the distance to the nearer points. This is recommended.

A minimum number of points is needed to straighten a segment. If insufficient points are available, then nothing will be done.

Straight fitting is done by finding the endpoints of the straight segment using the provided distances. Only existing points will be used: no point interpolation will be done. Then for points between these endpoints, a projection operator is used to map the new point onto the segment connecting the two endpoints. The original data should be close to straight so this projection operation does not result in any retrograde motion. This would result in 180 degree turns. So if the points are nearly straight, and the end points are good, this can help straighten out sections of road which should be perfectly straight. The points are moved onto the projected position on the line segment, but the elevations (or other parameters) are not changed. So the assumption is that moving onto the straight line is perpendicular to any altitude gradients. This could result in a road with a constant gradient but following a curvy path ending up with a non-constant gradient. So check altitudes with care. One possible approach is to use the `-uniformGradient` option to interpolate the altitudes, after doing the straight-line fit.

**`-straightEnd` <meters> (1 or more)**

Set one or more distances for straight fitting.

See `-straightStart` and `-straight`.

**`-straightStart` <meters> (1 or more)**

Set one or more start distances for straight fitting.

If multiple values are listed, then multiple sections will be replaced. Distances will be updated after each segment replacement. So if you want to do multiple circular replacements, list them from further to closer relative to the start to avoid the distances for later fits being affected by earlier fits.

See `-straight` for comments.

**`-stripSegments`**

if specified, this results in existing segment definitions on input files (including those specified with `-join`) being stripped before processing. This would typically be used if

---

`-autoSegments` is specified.

**-title <string>**

Specify the name of the GPX route. The default is the name listed in the source GPX. This is synonymous with `-name`.

**-track <number>**

Specify which track (1, 2, 3, ...) within the file to use. The default is 1. If you specify 0 or a negative number, the default will be used. Track number can also be specified by putting a suffix after the filename, for example "file.gpx:2" for the second track in file.gpx.

**-uniformGradient <start-meters> <end-meters> <bow-meters> ...**

This is similar to `-flatten`, except instead of setting a fixed altitude at each endpoint, interpolates altitudes from the course. One common application is bridges or tunnels, where a mapping program may use altitudes from the Earth, while the bridge or tunnel more typically takes a uniform gradient. Suspension bridges, for example, may be more parabolic in nature, and that will require future support. This is done before most route processing, but after `-join` and `-splice`, and after `-reverse`.

The third number, if any, is a bow term, which is how much the center point is above or below the straight line connecting the start and finish point. This results in a uniformly graded gradient rather than a uniform gradient. It is useful for suspension bridges, for example, which have a curved trajectory.

**-v or -version**

Print the version number and exit.

**-xShift <meters>**

Shift longitudes to shift the course eastward by this distance

**-yShift <meters>**

Shift latitudes to shift the course northward by this distance

**-zAutoSmooth**

Synonym for `-autoSmoothZ`

**-zOffset <meters>**

This is like `-zShift`, but is applied *before* scaling.

**-zScale <factor>**

Multiply altitudes in the original file *after offset*. but *before shift*. This is useful for fantasy routes where climbing should be adjusted, or potentially for data from a poorly calibrated barometer where I want to perfectly tune the net altitude change of a climb.

**-zScaleRef <meters>**

This is the reference altitude which will not be affected by `-zScale`. So if you have a course which starts at 100 meters and extends to 600 meters, but it should go from 100 meters to 700 meters, you would specify `-zScaleRef 100 -zScale 1.2`.

**-zShift <meters>**

Add this to the altitudes in the original file. This is useful for "fantasy courses", or where altitude is recorded by an improperly zeroed altimeter. This is applied *after scaling*. So, for example, `-zScale 2 -zShift 100` will change an altitude 0 to an altitude 200, and an altitude 100 to an altitude 400.

**-zShiftEnd <meters>**

Distance on the route to end altitude shift. There will be a transition outside of this range. This is also used for `-zScale` and `-zOffset` (it affects each).

**-zShiftStart <meters>**

Distance on the route to start altitude shift. There will be a transition outside of this range. This is also used for `-zScale` and `-zOffset` (it affects each).

**-zSmooth**

Synonym for `-smoothZ`

**-zSigma**

Synonym for `-smoothZ`

## EXAMPLES

### -auto option

The `-auto` option attempts to use "reasonable" parameters which may not be the best in each case, and which may require some fine-tuning, but should work fairly well:

```
processGPX -auto GPXData.gpx
```

This will create an output file "GPXData\_processed.gpx" with various options automatically chosen.

### criterium course

The following example was from a criterium course:

```
processGPX -laneShift -3 -shiftStart 740 -shiftEnd 1390 -spacing 3
-autoSpacing -cornerCrop 6 -minRadius 6 -prune -smooth 7 -snapDistance 2
-snap 1 -closeLoop CherryPieCritRGT.gpx
```

The course has an out-and-back section ending in a loop. In the actual race, the out-and-back is separated by cones. But Biketerra does not have a way to specify riders have riders go out and back on the same road except for full out-and-back courses. So here a separate road out and back are used. This creates four lanes.

Options:

**-laneshift -3**

Riders remain to the left left on the out-and-back portion, so each direction is shifted 4 meters to the left. The negative number implies left, a positive number implies right.

**-shiftStart 740 -shiftEnd 1390**

The lane shift is applied starting at 740 meters and ending at 1390 meters, with a transition calculated from the lane shift. This is applied after all smoothing, but before the lane shifting, and importantly, before the route start shift. Note there is a default `"-shiftSF -60"` applied at the end of the process, so if you want to determine the distance to apply the lane shift, then make sure to either do a run first with `"-shiftSF 0 -laneShift 0"`, or subtract 100 meters from the distance coordinates to judge what the distance would have been at the time the lane shift is applied.

It's important to check to make sure at the edge of the lane shift region the two directions don't get too close, due to the transition. If they do, then extend the lane shift region somewhat to make room for the transition. Also check the lane shift hasn't caused any corners to fold into points, or invert. If this happens, apply more smoothing to round the corners more before applying the lane shift. Unfortunately there is no support yet for position-dependent smoothing, which would help.

**-spacing 3**

This is a short criterium course with a lot of tight corners, so the initial spacing between points is set to 3 meters. This works for the 1.6 km criterium, but would perhaps be too many points for a 75 km point-to-point course, for example. But for longer routes, sharp corners are probably less of a factor. Note this number should probably be no greater than the `-smooth` parameter, unless `-autospace` is used, in which case corners will set with finer spacing.

**-autoSpacing**

This option is almost always a good idea, setting the spacing in the corners to provide adequate resolution.

**-cornerCrop 6**

If there are tight corners in the GPX file, this replaces the rounded corner with a circular arc before further processing.

**-minRadius 6**

Make sure the corner isn't too sharp for BikeTerra.

**-prune**

Remove useless points which don't affect either the shape of the route, or the altitude profile. This should probably be the default.

**-smooth 7**

Position is smoothed with a Gaussian with  $\sigma = 7$  meters. This was done here to tune the corner rounding, especially since there was a lane-shifted corner, and lane-shifting reduces inside corner radii.

**-snapDistance 2 -snap 1**

Snap the return leg in the out-and-back to match the outgoing leg when the two are within 2 meters. The "1" refers to replacing later occurrences of road with preceding cases: "2" would replace the earlier occurrence. 2 meters is presently the default snap distance. With Strava Route Editor data, 1 meter can result in a fragmented replacement, and bad results. When a road has curves, a larger snapDistance than the default may be necessary. Too large a value may cause merging roads to suddenly "snap" together from too far a range, however, or even adjacent roads or lanes to merge. For reference, in RGT Cycling, road width is 8 meters.

**-closeLoop**

It is a multi-lap race, so assure a smooth transition from the end of a lap to the beginning of a next. Also processing like smoothing recognizes that the course passes through the S/F point.

**out-and-back course**

This example uses the -outAndBack shortcut to create an out-and-back course starting from a simple point-to-point GPX file:

```
processGPX -outAndBack outSection.gpx -out outAndBack.gpx
```

**-outAndBack**

This specifies that the route should loop around at the turn-around, then return, with a default lane shift.

**out-and-back circuit**

This example uses the -outAndBackLap shortcut to create an out-and-back circuit with loops at both ends to create a circuit.

Point-to-point routes on BikeTerra have automatically generated turnarounds at the end so this is not normally needed there, unless for example you prefer having larger-radius turnarounds.

```
processGPX -outAndBackLap outSection.gpx -out outAndBackCircuit.gpx
```

**-outAndBackLap**

This specifies that the route should loop around at the turn-around, then return, then loop again, with a default lane shift.

## road course w/ out-and-back

This is from a road course with an out-and-back section. Note `-rUTurn` is not used here, since in this example, the loop at the end of the out-and-back is part of the original route.

BikeTerra automatically adds turnarounds to the end of point-to-point routes. so this is not typically necessary.

```
processGPX -crop 38730 -anchorSF -spacing 10 -autoSpacing -smoothAngle 20
-prune -smooth 10 -smoothZ 20 -snapDistance 5 -snap 1 -autoSegments 10 0.5
-cornerCrop 6 -minRadius 6 NoonRide.gpx
```

### **-crop 38730**

The finish of this route is on the out-and-back section, so alignment of the outward and inward legs is important. Since smoothing at the end of a GPX file is different than smoothing sufficiently far from the end, and the desire was to have smoothing similarly affect the out and back portions at the eventual finish, the GPX file was designed to extend beyond the ultimate end of the file, then it was cropped back. Cropping is done after smoothing, so this cropping distance was chosen by first examining the result of all smoothing, then processGPX was re-run on the original file with this crop distance added.

### **-anchorSF**

Don't move the start point or the finish point of the course. Smoothing is done as normal, but then at the end, these points are returned to their original positions, and nearby points nudged to keep a smooth transition.

### **-spacing 10**

A point spacing of 10 meters is initially established. This is more than what was used in the criterium course example, since for a longer course, smaller spacing results in more points.

### **-autospadding**

Automatically put extra points near corners before smoothing. This is a good option to assure smooth corners.

### **-smoothAngle 10**

Target the angle between segments at the apex of corners to be no more than 20 degrees. This seems to work fairly well. You can compare to 10 degrees. Most of the corner will end up with smaller angles than this, as will corners which are less than 90 degrees.

### **-prune**

Eliminate unnecessary points at the end. This should probably always be used.

### **-smooth 10**

Use 10 meter smoothing on position and, initially, on altitude. This results in some rounding of corners. For this course the result was compared with the "GPX Visualizer" website to satellite data, to make sure corners were fairly well aligned with actual corners, but additionally that there were no anomalies such as "zig-zags" which did not exist in the real road. More than 10 meters and some detail from the actual road may be lost, such as switchbacks with imperfect variable radius.

### **-smoothZ 20**

Additionally smooth altitude with a 20 meter smoothing distance. On this course, there were still gradient spikes with 10 meter smoothing, while more than 20 meter smoothing would have lost some of the actual variations in steepness.

### **-snapDistance 5 -snap 1**

The course is a "lollypop", meaning it heads out, does a big loop, then returns (part way). To make sure the return is well-aligned with the out, snapping is used. "`-snap 1`" means align

the return to the out (rather than the reverse). "`-snapDistance 5`" means to snap points which are as much as 5 meters apart. 5 meters is a lot, and the result needs to be checked afterwards to make sure this doesn't result in transitions are too abrupt, but a large snapdistance can help make sure corners get snapped together.

**-autoSegments 10 0.5**

This specifies that segments should be generated for hills found in the route. The first number is the approximate altitude gained in meters needed for a hill to be marked as a segment. The second is how strongly average gradient should be used in segment placement: a higher number places a higher value on segments having a higher average gradient.

**-cornerCrop 6**

It is generally good practice to set `-cornerCrop` to match `-minRadius`, to replace sharp corners with circular arcs before further processing.

**-minRadius 6**

If a corner turns out sharper than 6 meters, RGT will try to increase the radius of the turn by extending the road outward, with a smooth transition into and out of the correction.

**NoonRide.gpx**

This is the name of the original file. The processed file will be *NoonRide\_processed.gpx*. If the result is good, it's best to rename this to something different, so if you rerun the `smoothGPX`, it doesn't get over-written.

## simplifying

This example applies the automatic settings but then "simplifies" the points.

```
processGPX -auto -simplify original.gpx -out processed.gpx
```

**-auto**

Apply the automatic set of options for general smoothing.

**-simplify**

Apply an algorithm reducing the number of points based on a match to interpolated positions and altitudes. This is generally more aggressive than `-prune`,

**original.gpx**

This is the name of the original GPX file

**-out processed.gpx**

This specifies the name of the file to which the result is written

## selective smoothing

Here is an example where an urban route with reasonably sharp corners, except it followed an oval path around a park. The oval path came out of Strava Route Editor slightly ragged, so I wanted enough smoothing there to make it smooth, but the rest of the route, I wanted less smoothing.

One approach here is to use the `-selectiveSmooth` option. For example:

```
processGPX -closeLoop -spacing 3 -zsmooth 10 -selectiveSmooth 15 270 5 1540  
15 original.gpx -minRadius 6 -out processed.gpx
```

**-selectiveSmooth 15 270 5 1540 15**

This says to smooth with a smoothing length up to 270 meter distance, then 5 meters from there up to 1540 meter distance, and from there 15 meters again to the end of the lap.

## multi-step processing

But an alternate approach is to run the code twice, smoothing in two steps.

For this I used `-smoothStart` and `-smoothEnd`, to isolate the oval, but then to get smoothing on the rest of the loop as well, I needed to run the code twice:

```
processGPX -shiftSF 0 -lap -spacing 3 -zsmooth 10 -smooth 5 original.gpx
-out - | processGPX - -closeLoop -smooth 15 -prune -smoothStart 1540
-smoothEnd 270 -minRadius 6 -out processed.gpx
```

This uses a shell "pipe", which is a way to run the program twice on the same data without saving to an intermediate file. although the intermediate file would be useful for debugging.

### **-shiftSF 0**

The first call to processGPX specifies no S/F line shift (that will be done the second call, and I want to maintain the position of the start of the GPX for the first call).

### **-spacing 3**

A fine spacing is used here, as the loop is only 1600 meters, and I'll rely on pruning to reduce the number of points later.

### **-zsmooth 10**

This is moderate altitude smoothing. It's an urban course, but the climbs have fairly smooth transitions, and observing the gradient profile, there were some anomalies with using 5 meter smoothing. Strava Route Editor tends to produce abrupt gradient changes.

### **-smooth 5**

This is a fairly small amount of smoothing, for urban corners with some rounding, or where the actual road is wider than the Magic Roads 8 meter road width, and wider lines are available.

### **original.gpx**

This is the name of the original GPX file

### **-out -**

This tells the code to write the resulting GPX to "the standard output".

### **| processGPX -**

This tells the command line shell to "send the standard output to the standard input of the next program", which is a separate call to processGPX. It's called a "pipe". The "-" tells the code that this call takes its input from the pipe. So think of it as a virtual file, a direct line of communication from one call of the code to the next.

### **-closeLoop**

Specify this is a lap course with the last point coincident with the first. Copying the point may be redundant.

### **-smooth 15 -smoothStart 1540 -smoothEnd 270**

Apply 15 meter smoothing this time, except start it at 1540 meters into the course, and end it as 270 meters into the course. The smoothing domain wraps around, because it starts after it finishes. So the end of the loop, and the beginning of the loop, will be additionally smoothed, while the rest will be kept unaltered, with a transitional range applied to avoid abrupt changes.

### **-minRadius 6**

This sets the minimum radius of corners to 6 meters. This is done in the second step here to include effects from processing in both steps.

### **-out processed.gpx**

## adjusting gradient for bridges or tunnels

Typically tunnels are built with a constant gradient. Route editors may not produce this uniform gradient, so it can be useful to "manually" set the gradient to uniform. Or, on climbs with poor altitude resolution, you might want to set the gradient to uniform over certain sections.

Here the `-uniformGradient` option can be useful, for example:

```
processGPX -auto -simplify -uniformGradient 1000 1500 input.gpx -out
output.gpx
```

Options here include:

### **-auto**

This specifies a reasonable set of smoothing parameters.

### **-simplify**

Reduce the number of points after processing beyond that which `-prune` would produce.

### **-uniformGradient 1000 1500**

This specifies that the gradient from 1000 meters to 1500 meters is to be set to be uniform. This is done before any other processing, so additional smoothing may transition the gradient at the start and finish of the interval.

For suspension bridges, for example, a uniform gradient might not be appropriate. These may have a smoothly varying gradient, producing a parabolic shape. In this case, you would provide a third parameter to `-uniformGradient`. For example:

```
processGPX -auto -simplify -uniformGradient 2000 2400 3 input.gpx -out
output.gpx
```

### **-uniformGradient 2000 2400 3**

This specifies that instead of a uniform, linear increase in gradient from the start at 2000 meters to the finish at 2400 meters, the center is raised 3 meters relative to the straight line, with the transition to 0 meters at the start points and stop points.

## segment generation

```
processGPX -stripSegments -autoSegments 100 0.5 -segments 1500,2000,"bonus
segment" input.gpx -out output.gpx
```

### **-stripSegments**

If the input file has segments already defined, this will ignore those.

### **-autoSegments 20 0.5**

This tells the code that automatic segments should be generated for climbs gaining at least 20 meters if they average 10% gradient. Shallower climbs need to gain more, steeper climbs don't need to gain as much. The power assigned to gradient is 0.5, so for example if the gradient is 5% instead of 10%, then since the square root of  $5/10 = 0.71$ , this climb would need to gain at least 28.2 meters instead of 20 meters to be considered as a timed segment.

### **-segments 1500,2000,"bonus segment"**

This adds an additional segment to the route. between 1500 and 2000 meters from the start of the route. This is defined before any automatic segments are generated, so care should be taken that this segment is not overlapping anything which will be considered to be a climb.

```
processGPX -stripSegments -autoSegmentMargin 1000 -autoSegmentStretch 1
-autoSegments 5 1 input.gpx -out output.gpx
```

This example uses a gradient power of 1, placing a large emphasis on local gradient, but then increases the auto-segment stretch factor from its default to 1. This might be useful if I had a long



climb with intermediate descents along the way, and I wanted separate climb segments for each climbing portion rather than the net climb, but I wanted the endpoints of these segments to be on flat road if that was at all possible within the constraints of the margin, which is set to 400 meters.

**-autoSegmentMargin 1000**

The minimum spacing between auto-segments is set to 1000 meters. The minimum spacing to the start line would be set with -autoSegmentStartMargin, and for the finish, -autoSegmentFinishMargin.

**-autoSegmentStretch 1**

Be willing to increase auto-segment lengths up to their original length in order to either reach a flat portion for the start or a flat portion for the finish,

**-autoSegments 5 1**

The climb threshold is set to a relatively low value of 5 meters, meaning I want to pick up even small steps in the larger climb, and the gradient factor is set to 1, putting an emphasis on isolating individual climbing portions.

## adding time to an activity

Suppose I wanted to add a time field to the result of the preceding example, because I want to upload the GPX to "Relive.cc" so I can generate an animation of the route to include in an event description of a race I'm organizing on the course.

```
processGPX -startTime "Feb 25 2021 07:00" NoonRide_processed.gpx
```

Here I am telling the code to use its bike speed model to predict how long it will take a relatively fast rider to reach each point of the route, and to add a time (and "duration") field to the GPX file, which will be accepted by the RideWithGPX website. The resulting file will be *NoonRide\_processed\_processed.gpx*.

**-startTime "Feb 25 2021 07:00"**

This specifies that the time points begin on the listed date and time in the local time zone (local to the user, not the course). The format of the date and time are flexible, but try to be unambiguous. For example, rather than put "01/02/03" for a date, try "02 Jan 2003".

## adding gradient signs

This is an *experimental* feature since RGT, the game which is the primary target of this code, does not at present include waypoint support.

The following shows a partial command line, so added to other elements of a command line:

```
-addGradientSigns -gradientThreshold 20 -gradientPower 2
```

**-addGradientSigns**

Tells the code to add waypoints where gradient signs should be placed.

**-gradientThreshold 20**

A 10% grade would need to gain or lose 20 meters to get a sign. The altitude required for other gradients depends on the next option.

**-gradientPower 2**

This is the default value, but is listed here for documentation purposes. It says the suitability of a climb for a gradient sign is proportional to gradient squared. So for example, if a 10% climb gets one if it climbs 20 meters, then a 5% grade would need to gain 40 meters. This also affects the placement of signs, since if a climb is gradual, then steeper, then gradual again, should a single sign be used to cover the entire climb, or should signs be prioritized to the steep portion, then possibly add additional signs to the gradual portions if they meet the threshold? The higher gradient power, the greater the priority placed on steepness. The

default of "2" seems to work well.

### processing a BikeTerra route from the original GPX

Suppose you have a BikeTerra route but don't have access to the original GPX, and you want to refine that GPX file to improve corners. You might have made some edits on the route editor, but you don't care about these.

Here is how you might do that:

```
processGPX -auto BikeTerraGPX:12426 -out route.gpx
```

Or, if the data were from Strava, you might want to add `-fixSteps`:

```
processGPX -auto BikeTerraGPX:12426 -fixSteps -out route.gpx
```

This will download the original GPX file from route 12426 and create a GPX with the `-auto` option applied.

Alternately, maybe you just want to download that file with minimal change. Then you would leave out the `-auto` option in this example. It will check for zig-zags, for example, and little more.

```
processGPX BikeTerraGPX:12426 -out route.gpx
```

### processing a BikeTerra route from route data

Suppose you have a BikeTerra route and maybe you've done some edits with the route editor, but since it's hard to create smooth corners with the GUI, you want to do some final smoothing before uploading a revised GPX file. You could do that as follows:

```
processGPX -auto BikeTerraRoute:12426 -out route.gpx
```

If you simply want to download a GPX representation of the route, you can leave out the `-auto`:

```
processGPX BikeTerraRoute:12426 -out route.gpx
```

### specifying metadata

GPX files have "metadata" which is various tags. You can change values of metadata with various options. This is an example:

```
processGPX -author "Dan Connelly" -keywords "race, BikeTerra" -copyright  
"Dan Connelly" -name "Crit Course" crit.gpx -out critBT.gpx -description  
"the best crit course"
```

This example specifies an author name, adds keywords, a copyright, a title, and a description, taking the trackpoints from the file "crit.gpx", and writing the result to "critBT.gpx". The time the file was generated is automatically stored in the "time" metadata field. A "processGPX" keywords is additionally automatically added, to record this program was used,

### converting an RGT route to BikeTerra

RGT routes had 60 meters at the start and 140 meters at the end for riders to group. Biketerra does not use these buffer regions. So to get the same course distance in BikeTerra as was used in RGT, 60 meters from the start and 140 meters from the end should be cropped. For example:

```
processGPX -cropMin 60 -cropMax -140 RGTRoute.gpx -out BikeTerraRoute.gpx
```

For circuits, you don't need to crop the end of the route, but the S/F point should be shifted:

```
processGPX -cropMin 60 RGTCircuit.gpx -out BikeTerraCircuit.gpx
```

### NAMED SEGMENTS

The code has been enhanced to handle "named segments" as was interpreted by RGT. Segments are defined in the GPX standard such that a "track" consists of a series of "segments", and a "segment" consists of a series of "trackpoints". In RGT, segments are assumed to be contiguous,

such that a route is defined as the first segment, connected to the second segment, connected to the third segment, etc.

RGT no longer exists, but this code is retained since there is a chance another game will implement similar functionality. If the details differ, then this code will be updated to accomodate that. If anyone knows of any other games which support a similar function, let me know.

Segments can be either named or unnamed. Named segments are interpreted by RGT to be timed separately, similar to Strava segments, where times it takes riders to go from the beginning to end of the segment are recorded. This program interacts with segments in several ways:

1. if the input file has segments defined in the manner required by RGT, then these segments will be retained, unless the `-stripSegments` option is used. If there are gaps between segments, then these gaps will be filled with additional segments, such that there is no ambiguity about whether a gap belongs to the prior or following segment.
2. Adjacent unnamed segments will be merged, so for example the artificial segments assigned to gaps between segments will be assigned to adjacent unnamed segment(s).
3. You can define your own segments using the `-segments` option. This is followed by a comma-delimited list of a start position, a finish position, and a name, followed optionally by more of these three items. So it can be followed by a number of comma-delimited elements divisible by three. For each triplet, a segment is formed between the first coordinate and the second coordinate and assigned the name of the third element (if non-blank). For example, `-segment 1000,2000,"timed km"` creates a segment from 1000 meters into the route, up to 2000 meters into the route, and named the segment "timed km". These coordinates are of course affected by other operations performed by this code so order of these operations is important. The segments are evaluated after most operations such as smoothing, but before cropping, so the coordinates do not take into account potential crops.
4. A more powerful way to create segments is to do so automatically. This uses an algorithm which examines the route profile and identifies what it considers to be "climbs". The default is for no automatic segments to be generated, so to request this, the `-autoSegments` option is used. It's followed by two numbers. The first is the threshold for a climb: small bumps in the road are not assigned for timing. The number is vertical meters at a reference 10% grade. The second number is a power which describes how much weight is put on steepness for defining a climb. A typical range is 0.3 to 3.0, with 0.5 to 1.0 working fairly well in many cases. A lower number will tend to create longer segments rather than focusing on steep sections, and will tend to combine sections which are separated by plateaus or brief descents. A larger number will tend to split these segmented climbs into individual climbs, so for example if a road is 10% for 1 km, then flat for 500 meters, then 8% for the next km, that might be considered one climb or two. Additionally automatic segments need to be separated from each other. The amount of this separation is specified with the `-autoSegmentMargin` option (default is 400). There's also minimum spacings to the start (`-autoSegmentStartMargin`) and finish (`-autoSegmentFinishMargin`) lines. If climbs would be too close to each other, the lower rated one is ignored, as are climbs which would be too close to the start and/or finish. So if you're designing a route for a hillclimbing competition, unless the finish is sufficiently past the start of the climb and the start is sufficiently separate from the start of the climb, there will not be a segment defined for the climb.

## BUGS and ISSUES

### lane shifting and sharp corners

Lane shifting with sharp corners can cause issues, so should always be combined with sufficient smoothing. If you apply line shifting, make sure there is sufficient smoothing.

### `-snapTransition` may cause issues.

If there are gaps in a snapping range, then `snapTransition` may cause unnecessary flattening of the course

### snapping and segments

If segments are defined before snapping, the code will try to assign the appropriate segment to each snapped point, but this is imperfect. It is better to define segments after snapping points. This is unimportant for Biketerra, which ignores segments.

## **AUTHORS**

Daniel Connelly <*djconnel@gmail.com*>

## **LICENSE**

This application is free software; you can redistribute it and/or modify it under the same terms as Perl itself.