NAME

processGPX

SYNOPSIS

smoothGPX [options] <input files>

VERSION

0.40

DESCRIPTION

smoothGPX is a series of algorithms to improve and create GPX files, in particular for cycling emulation platforms like RGT Cycling.

At the time of this writing, RGT Cycling has a "Magic Roads" functionality whereby a GPX file can be submitted for conversion into a virtual envirment. 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, RGT 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. Also, RGT at present allows cyclists to use the full width of the roadway, even if there is an out-and-back section where there may be oncoming riders. So it is useful to allow the generation of out-and-back courses where the return route is shifted to the right or too the left, to avoid the possibility of virtual collisions.

These are just some examples of the sort of processing which can be done to improve the quality and functionality of Magic Roads in RGT, and presumibly 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.

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.

The program will calculate a "altitude quality score" of both the original GPX file, and of the result of processing, which is based on how much the grade changes point-to-point -- it is basically the ratio of the root-mean-squared change in gradient to the root-mean-squared average gradient for each point in the course. The goal is to only have abrupt gradient changes where they actually exist in the real-life course. A score of 1 on this metric is poor: it means there are many abrupt gradient changes. A score of 0.1 is smooth: gradients typically change around 10% from segment-to-segment. This quality score can provide guidance to if altitude smoothing and point density are sufficient. This doesn't address the issue of whether the "top-down" view of the course is optimally smoothed. For

this careful examination of the resulting GPX on GPX Visualizer, or simular, with "aerial view" is a good idea.

This program has been enhanced with the ability to process "named segments", which RGT interprets as sections of a course which should be separately timed. A typical application of these is for climbs, and the program allows the auto-generation of segments using the detection of distinct, non-overlapping climbs in the route.

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 comamnd-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
```

OPTIONS

The program is generally invoked:

processGPX [option] <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, -nosave suppresses saving of the resulting GPX in an output file.

* **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**: the "-fixed" option is followed by a list of one or more numbers. This will be explained in the section for that option.

The options are the following:

addDirection

This option adds a "direction" 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.

addCurvature

This option adds a "curvature" field (in 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 reciprical of the circle radius.

addDistance

This option adds a "distance" field (in meters) to the GPX file. Distance is calculated after all other operations, for example cropping, extending, and shifting. At present it's the horizontal distance, so scaling a vertical cliff would count as zero.

addGradient

This option adds a forward "gradient" field, which is calculated for each point as the ratio of the altitude change to the horizontal distance to the next point. With RGT, this is useful for checking whether altitude smoothing is sufficient.

addGradientSigns

This flag is an i<experimental> option to calculate regions of exceptional gradient and place "gradient signs" in the GPX as waypoints. RGT Cycling at present ignores waypoints, but perhaps support will be provided in the future, in which case this feature will need to be tuned to the appropriate specification.

addSigma

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

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 is typically places 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 is 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 this case.

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

Since RGT 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 is useful for preventing RGT from connecting the finish and start points, which has a 100 meter threshold, as this is written.

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. Altitude auto-smoothing is used if no value for -zsmooth is provided.

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 (gradient / 10%)^power, where the default power is 0.5. The names can be set with "autoSegmentNames", with a generic default.

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.

autoSegmentMargin <meters>

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

autoSegmentFinishMargin <meters>

A minimum buffer between an auto-generated segment and the finish banner. This defaults to 50 meters to allow room for riders to see a finish banner after passing the segment banner.

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.

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 2, depending on how much focus you want to put on steepness in defining and rating climbs.

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)

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 "-sigmaz" 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. =item **closed**

Synonym for "-copyPoint"

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 ethe further points are listed first so the fit of a circle to these points does not affect the distance to the nearer points.

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 circlar 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 you need to run processGPX more than once. Note the circular arc will slightly change distances so you may want to do the corrections starting further from the start, progressing closer, for more stability in these distances.

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, either list them from further to closer relative to the start to avoid the distances for later fits being affected by earlier fits.

copyPoint

For "-loop" courses, copy the first point to the end of the list of points, so the circuit is explicitly closed. This assures that RGT Cycling will recognize it as a loop course.

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.

crop <meters>

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

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 sufficiently ahead of and behind the finish so that smoothing has access to similar points both on the final and on preceding passes of that section of road.

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.

crossingAngle <meters>

If set, then this is the minimum degres 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 minumum 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.

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 specied with either a "csv" or "gpx" suffix (case insensitive), in which case the suffix is used to determine the format. Other suffixes are ignored.

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.

disableElevationFixes <0 or 1> =item **disableAdvancedSmoothing** <0 or 1> =item **enableElevationFixes** =item **enableAdvancedSmoothing**

These options specify whether to explicity turn off or explicitly turn on various RGT post-processing options. These functions are provided by RGT, not this program, which simply places the appropriate tags in the GPX output. Note the "disable" options take a number as an argument: 0 or 1. the "enable" options do not take a number argument. So this provides two ways of specifying whether you want these features. Enabling supercedes disabling.

The default is to disable both.

extend <meters>

This is a simultaneous specificiation of both -prepend and -append.

extendBack <meters>

With RGT point-to-point Magic Roads, the game will reserve the final 140 meters for riders finishing. If you've created a route to the top of a climb, for example, you may want the riders to be able to ride all the way to the top, rather than stopping up to 140 meters before the top.

One way to deal with this is to use -append: this will extend the route with a straight line at the finish. This may be enough, but this 140 meters might not be consistent with the real topography, for example the top of a twisting climb.

So this is an alternative: it will create a loop of radius specified by -rTurnaround, then start returning on the route until the specified distance, *including the loop*, is covered. If the loop is large enough, the route may end on the loop itself. If not, the route will end on the return section.

It works by doing a normal turnaround, then cropping the course. So if there's not enough course to reach the specified distance, it will turn the route into a full out-and-back. But typically the distance should be in the range of up to 140 meters or so.

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 cop`ies 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.

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.

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.

flatten <start-meters> <altitude-meters> <end-meters> <altitude-meters> <transition-meters> ...

This important 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 sepecify 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, yet small enough to limit the extent of the influence of the command.

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

Distances are calculated i<after> smoothing but before cropping or extension.

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 altitude gained 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, segment found in these files and append them to the first track, segment in the original file. The files must reasonably match up end-to-end.

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.

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.

maxSlope <%>

RGT supports 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.

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.

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. This is done very late, for example after -rUTurn.

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.

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.

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.

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.

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, so for example a 45-degree include has gradient = 1), then the point B is not 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, althouugh 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.

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 an 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".

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 turn The main turn will be to the left if laneShift is negative (UK, for example), or to the right if laneShift is positive (US, for example) with a default (for no laneShift) to the right.

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.

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 commas: 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 these options with a loop course, since for loops, the course will not start or finish at the GPX end. In any case, 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.

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

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 regaions 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.

shiftSF <meters>

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

For loop courses, RGT Cycling puts the start/finish line 60 meters after the start of the GPX data. This is evidently to provide room for riders to line up before the start, although that is only necessary in point-to-point courses, so the reason for this is unclear. In circuit courses careful placement of the start/finish line, for example at the crest of a hill, or a specific distance from a final corner, is critical, so with this option you can carefully place the start point for the GPX file at the desired start/finish, then use "-loop -shiftSF -60" to move the start of the GPX file back 60 meters, putting the start where it should be.

Since this is so commonly used in loop courses, the default is -60, so to disable it, use "-shiftSF 0 -loop".

sigma <meters>

Synonym for "-smooth"

sigmaz <meters>

Synonym for "-smoothz"

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.

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.

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 distace 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 oup 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.

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>

WARNING: this may be buggy. Check results carefully.

This is an i<experimental> feature whereby points within this distance of a snap transition on partially syncronized in altitude, with a transition weighting depending on distance along the course. This is designed to avoid having divergent roads having different slopes while still overlapping, which can create "ridges" in the pavement which in real life would cause riders to crash. 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. -autospacing is another option, in which case the code will selectively interpolate points near points where the direction is changing.

splineDegs <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 -splineMaxDegs 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.

splineMaxDegs <degrees>

If a -splineDegs 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: verty strange results can occur if the corner is too sharp.

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 an 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

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.

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 heureistic formula which has rider speed depend on the road grade.

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. Not ethe further points are listed first so the fit of a straight to these points does not affect the distance to the nearer points.

straightEnd <meters> (1 or more)

Set one or more distances for straight fitting.

See -straightStart

straightStart <meters> (1 or more)

Set one or more start distances for straight fitting.

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 staight 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 -flatten option to interpolate the altitudes, after doing the straight-line fit.

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, either list them from further to closer relative to the start to avoid the distances for later fits being affected by earlier fits.

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".

v or **version**

Print the version number and exit.

zAutoSmooth

Synonym for -autoSmoothZ

zOffset <meters>

Add this to the altitudes in the original file. This is useful for "fantasy courses", or where altiude is recorded by an improperly zeroed altimeter. This is applied *before scaling*. Use -zShift for changing altitude after 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.

zShift <meters>

Add this to the altitudes in the original file. This is useful for "fantasy courses", or where altiude is recorded by an improperly zeroed altimeter. This is applied *after scaling*. Use -zOffset for before scaling.

zShift <meters>

Add this to the altitudes in the original file. This is useful for "fantasy courses", or where altiude is recorded by an improperly zeroed altimeter. This is applied *after scaling*. Use -zOffset for before scaling.

zShiftEnd <meters>

Distance on the route to end altitude shift. There will be a transition outside of this range.

zShiftStart <meters>

Distance on the route to start altitude shift. There will be a transition outside of this range.

zSmooth

Synonym for -smoothz

zsigma

Synonym for -smoothz

SEGMENTS

The code has been enhanced to handle "named segments" as 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.

Segments can be either named or unnamed. Named segments are intepreted 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 1.0, with 0.5 to 0.7 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 tent 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 suffiently separate from the start of the climb, there will not be a segment defined for the climb.

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 -7 -shiftStart 740 -shiftEnd 1390 -spacing 3
-minRadius 6 \     -prune -smooth 7 -snapDistance 2 -snap 1 -copyPoint -lap
 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. However, for RGT cycling, riders use the full road width of 8 meters, so the out and back portions needed to be separated.

Options:

-laneshift -6

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.

At one time a lane shift of 4 meters was adequate, but with changes to "asset" placement, a larger lane shift is now needed. This example uses 7 meters to the left (UK, perhaps).

-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 "-autospacing" is used, in which case corners will set with finer spacing.

-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.

-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 Gaussin 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 occurances of road with preceding cases: "2" would replace the earlier occurance. 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.

-copyPoint

Make sure the last point matches the first, so RGT Cycling recognizes it as a circuit

-lap

It is a multi-lap race, so assure a smooth transition from the end of a lap to the beginning of a next. This is also necessary for "-copyPoint" to work.

road course w/ out-and-back

This is from a road course with an out-and-back section:

```
processGPX -crop 38730 -anchor -spacing 10 -autoSpacing -smoothAngle 20
-prune -smooth 10 -smoothz 20 -snapDistance 5 -snap 1 -minRadius 6
NoonRide.gpx
```

-crop 38730

The course was designed beyond the desired end of the RGT course, since the end is on an out-and-back section, and to assure the end is perfectly aligned with the outward portion of the same road, it was extended further and cropped back. This is important since smoothing uses both the road ahead of and behind a certain point, so at an endpoint of GPX data, smoothing will be different than if that road had been extended further. This course ended up being 38.73 km, which corresponds to a distance between banners of 37.63 km, since 60 meters (start) and 140 meters (finish) each end is reserved for riders gathering.

-anchor

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.

-autospacing

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

-smoothAngle 20

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.

-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.

Multi-step processing: 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. 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 - -copyPoint -lap -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).

-lap

It is a lap course.

-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 sepearate 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.

-copyPoint -lap

Tell this call to the code that it's a lap, and I want to copy the first point to the last point to close the loop.

-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

Multi-step processing: snapping after spacing

Snapping (making sure the course is aligned where it repeats itself) is performed before any point interpolation or smoothing. However, especially with short courses, it may not work well where straigh sections in the initial route have only two points, or elsewhere when the number of points is very sparse. A solution in this case is to run the program twice. This is an example:

```
processGPX -loop -copyPoint -shiftSF -50 -snapDistance 3 -spacing 2 -snap 1
 -smoothz 10  -lap CritRGT.gpx -out Crit_step1.gpx
```

```
processGPX -loop -copyPoint -shiftSF 0 -snapDistance 1 -snapTransition 20
-prune -snap 1 -smooth 10 CritRGT_step1.gpx -out Crit_processed_v1.gpx
```

Here the first pass does a first pass of snapping, and afterwards it adds points to a very fine spacing of 2 meters, It shifts the start of the GPX back 50 meters. The goal here is to do the first crude point snapping on the low-resolution data,then reduce the point spacing for a second snap pass which will be done next, since the code snaps only before adding points. The result of the first pass is the file *Crit_step1.gpx*.

The second pass is not going to shift the start of the GPX any further: that was already done the first pass. Gross snapping was done the first pass, so a smaller snap distance is used here, since with the finer spacing of points after pass 1, some additional snapping may be needed here. The experimental "-snapTransition" option is used here. It makes sure points entering or leaving snap regions are aligned in altitude. This was a reason for the very fine point spacing used here: this course has a lot of repeated sections, and the goal is to keep the roads at similar altitude as they converge or diverge to

avoid ridges in the asphault. Smoothing is done this step, with a 10 nm smoothing length. Note the snap transition was twice this value: we don't want smoothing to compromise our synchronization of altitude. The resulting file is *Crit_processed_v1.gpx*.

Some experimentation is needed with these options, but it helps to have an idea of what the requirements are of each specific course. The course shown here was special in taking multiple passes over the same roads (actually paths) and having a complicated altitude profile. This situation is particularly challenging with Magic Roads if you want the transitions to look good.

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 he 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 data and time in the local time zone (local to the user, not the course). The format of the data 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 i<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.

auto

This option will attempt to guess at some good settings based on the course.

-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, than 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 addiitonal 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.

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, RGT" -copyright "Dan
Connelly" -name "Crit Course" crit.gpx -out critRGT.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 reslt to "critRGT.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,

BUGS

lane shifting and sharp corners

If you apply more lane shift than the radius of the tightest corner, the corner could end up with a non-positive radius, which is not what you want. The code may in the future be enhanced to make sure lane shifting doesn't reverse the direction of any road segments. But if you want, for example, 4 meter lane shifting, then make sure smoothing is at least 5 meters.

snapping not perfect

The snapping algorithm makes certain assumptions to reduce computation time and can be conflused for sufficiently complicated routes. More testing is needed.

-snapTransition is suspect.

This option needs more debugging.

RGT ignores gradient signs, and puts in its own

This isn't the fault of this code: complain to RGT.

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.

AUTHORS

Daniel Connelly <*djconnel!at!gmail.com*>

LICENSE