

<http://home.online.no/~bi-knu/editing/default.htm>

<http://www.vanwall.eclipse.co.uk/gpleediting/>

GPL Track Creating

by the GPLEA (C) 2000,2001

intro

Updates:

1st release: 16-Feb-2001: Greetings ! Please note that the site is not yet completely finished :-)

Last updated 20-Mar-2001

Introduction

Creating a track for GPL is a loooong process. Do not expect to create something great in less than 2 months; something spectacular in less than 6 months or something driveable in less than 2 days...

So where do you start ?

- 1) Learn the ropes
- 2) Pick a track
- 3) Give up your social life, job, other-half
- 4) Spend weeks fixing every detail
- 5) Marvel in your creation being driven by thousands at Vroc

Introduction pt.2

There are several stages to creating a track, and you should be aware that they were not created by the same people, or in the same style, so you will have to learn a lot...

TRK/GTK (they are the same, but GPLTrk/trk23dow will only *load* GTK files).

Most people start with GPLTrk - the program written by Peter Prochazka to create .trk & .gtk files. Trk's contain the physical surface of the track, and are used as the basis of the graphical content.

3DO

When you have a .gtk file, you require a .3do file. trk23dow is the program used for this. It runs under good ol' MS-DOS, reads in several different files for texture information etc., eats serious CPUs for breakfast and outputs a nice pile of polygons wrapped up in a handy .3do file (or a bunch of errors, depending on what you gave it).

MIPs & SRBs

If you don't give trk23dow any details on the textures you want, it's going to use one default texture for

the whole track - not exactly what you want. Use your favourite paint program and one of WinMip, MipMan or SrbMan to make your track look like 1967.

3DOs (trackside objects)

Want a tree, bridge or bunch of loonies standing in the freezing cold watching bits of metal fly round a track? Add some trackside objects in the form of 3DOs. You have two options - use Paul Hoad's GPLTrackEditor, learn the art of Binary Space Partitioning and go mad staring at a bunch of vertices & invisible planes - or take the easy way out and use the Object Repository so kindly provided by fellow track creators. Then all you have to do is position them on your track so they don't clip with anything else (that word 'clip' is going to be your best friend, muahahahaaaa).

CAM

So you made your track, you go watch a replay, and all you can see is green or trees. All is not lost! Thanks to Joachim Blum of C.A.R.E., Track Cam has been created to let you position your cameras as you see fit.

PBFs

These are the program covers, track maps, weekend screen and newspaper pictures. You'll be wanting a paint program and WinMip/GPLBatch here.

LPs

Multiplayer is the way to go, but lots of people still race against the AI, so create some LP files from a smooth replay using Nigel Pattinson's rpy2lp program. One of the LP files is also used for shift-r, so you'd better make one if you don't want to get stuck in the armco every time you crash !

track.ini

Just some numbers, but they control a lot, so do it properly! Ed Solheim has written a tutorial.

The beginners 'guide' to making a GPL track PART 1

This tutorial/Guide is not very comprehensive, and, as such, leaves all the difficult parts of track creating drunk in a ditch. But hopefully it will get you going! The more scary items will be dealt with later. Share your knowledge! What we are going to create is a very basic track that will only use a few of these tools functionalities. Be warned; this is highly addictive, and you may end up without a social life! If you manage to have a few thoughts on your own apart from the usual distractions, this should get your started. The rest of the knowledge can be found around this site, on the forums, and if you send lots of dosh; via e-mail.

CONTENTS:

1: Prepare to Edit!

2: Creating Corners & Straights

3: Traces and Walls

4: Making textures Stick!

5: Adding an object, and going for a run

Note: A complete "mytrack", all done, is included if you wanna cheat. But who are cheaters? Well, okay, get [MyTrack.zip](#) here (it's also on the [Downloads page](#)).

Ok, I bet you are roaring to start! We need some preparation first, don't try to understand this yet, just do it!

Using windows explorer, create a directory for editing somewhere on your hard drive, for example **C:\GPLeedit**

In this folder, add another for your new track, let's call it "mytrack". So you have:

C:\GPLeedit\mytrack

From the [downloads page](#), obtain GPLTrk and trk23dow. Extract the following files into the "mytrack" folder:

GPLtrk.exe
trk23dow.exe
trk23dow.cfg

Now you'll need a few additional files which you make yourself using notepad. Open notepad (or any other text editor, I prefer UltraEdit- but keep it simple!) and type:

mytrack.tex

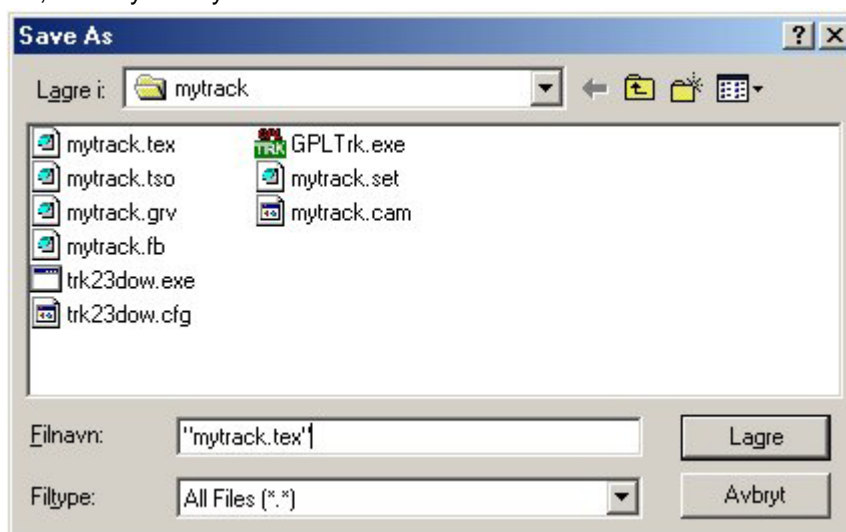
Save this file as "**mytrack.tex**" (not text, tex!) in the mytrack folder. To get the proper extension, you type the full name with speechmarks "mytrack.tex".

Close, and open a fresh notepad, type "mytrack.tso". Save with same name. Get it? Do the same procedure for

mytrack.set
mytrack.fb
mytrack.grv
mytrack.cam
mytack.ini

The files will be explained in a bit!


So, this is your mytrack folder:








Now you've got all the files required - almost! Now is the time to get some coffee, a pack of smokes (if used), chase the family away, and put some nice, soothing music on the stereo. If all goes to plan, you'll have a track (a very boring one, mind!) working in GPL in the not too distant future!


The beginners 'guide' to making a GPL track PART 2

2: Creating the twists and turns...








Now it all starts. From explorer, double-click  GPLTrk.exe and your screen is filled with.....lovely grayness! I suggest you make a shortcut to GPLTrk right away.

From the menu, choose **File, New**. Presto! You now see a length of gray, the red arrow pointing the direction of the track. This is your first section, and it defines the start/finish line, and a following straight which is 50 metres long. A short explanation of the top window:

Variable	Value	Meaning	Edit	Comment
 Length	984252	50.000002 m		
 Number of Sections	1			
 Number of Traces	2			
 Trace 0	-98425	-4.99999 m	yes	
 Trace 1	98425	4.99999 m	yes	

 **The length of this section**

You will not use normal numbers, but use decimals (listed in the Value column). You can always see the true meaning of this under...er "meaning". Now, 50m may be a long way to run, but not to drive, so lets up it a bit. The info you see now is just that; info. So you'll need to expand the tree on the left...this brings up:

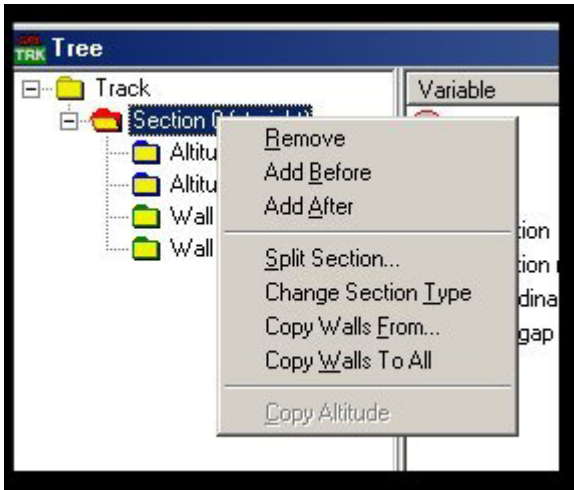
Variable	Value	Meaning	Edit	Comment
 Type	1	straight		
 Start	0			
 Length	984252	50.000002 m	yes	
 Orientation	0	0 °	yes	
 Orientation mismatch	0	0 °	yes	
 Longitudinal gap	-984252	-50.000002 m	yes	
 Lateral gap	0		yes	

Now, double click on "Length". Change the value from 984252 to 2984252. Your first section is now 151.6m long! (as a very rough guide, 20000 = 1 metre). Enough for a quick stab in 2nd gear! Now is perhaps a good time to Zoom out. You can do this by right clicking while holding the cursor in the "graphics area" (the lower 50% of the screen).

WARNING! Be careful when you zoom in, as GPLTrk may crash your PC if you zoom in too much. Try it for the hell of it, and you'll get the idea.....but maybe you want to save your track first:

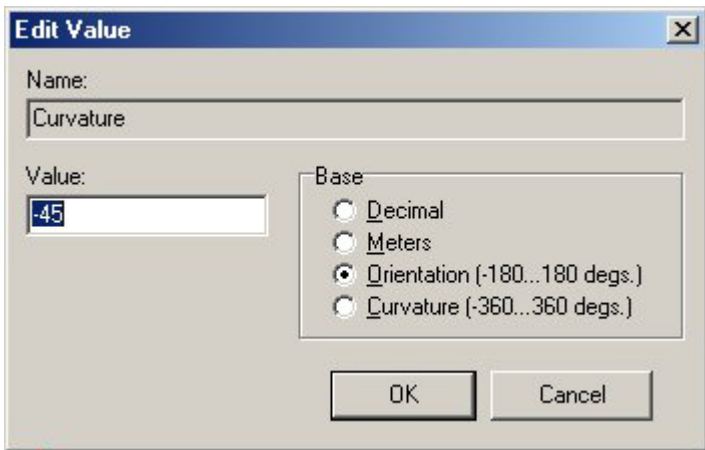
File-> Save As, "mytrack.gtk"

Ok, now we're gonna add a corner, to make things interesting. In the tree on the left, RightClick "Section0", and POP UP! Menu appears.



Now choose "Add after". By default, GPLtrk will now add a section identical to the previous, so you have another section of straight....Buuut, we want a right hand corner! To do this, right click on your new Section, and choose "Change section type". It will now curve gently left. I'm gonna keep this simple, now you have a host of other options on this sections since it is curved!

The first thing to notice; negative curvature means that the turn goes right. Double Click on "Curvature". In the box that appears, select the Radio Button for "Orientation", change this to -45 (degrees!). And click OK



Now you have a very long right hand bend, slightly jagged. Double click on Curvature again, and check that "Orientation" now displays -90.

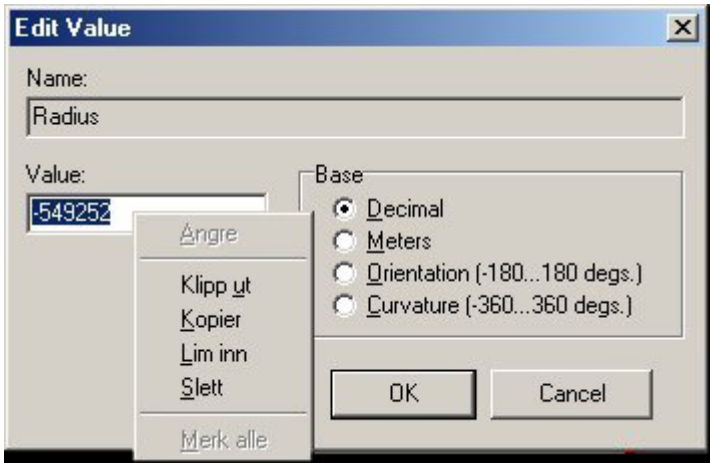
Now, Double Click "Radius". Check for "Decimal", and change -8549252 to -549252. Now you have a nice curve (a bit on the sharp side, but who cares!)

Now right click on the tree again, and select "Add After". Another section is added, a copy of your last turn. If all is well, you should now have a track looking like this:

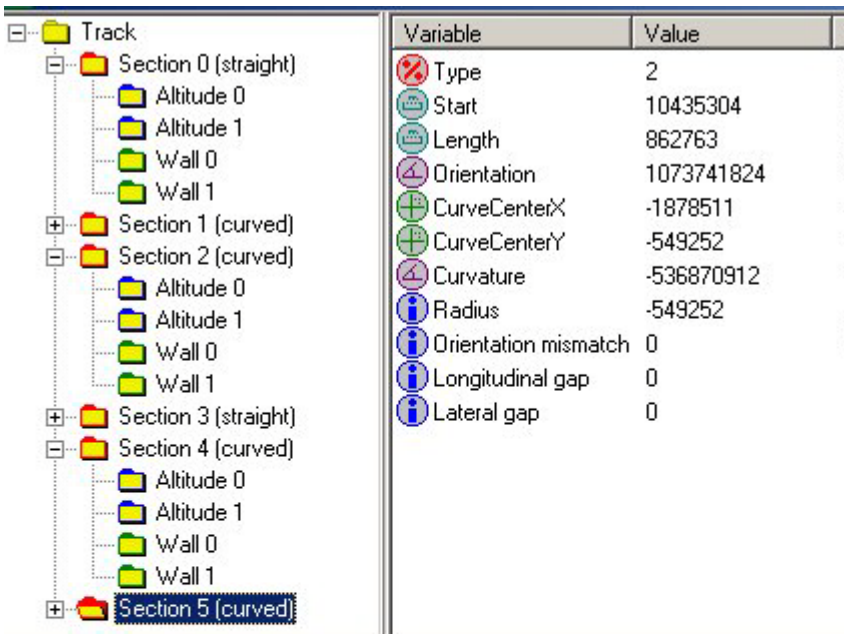


Still going? Good! Now click on "Add After" (you know the routine, right click on the last section, and....). Whoops, that turn now makes you feel dizzy! Er, so right click on that section again from the tree, and Change Section type. You now have a small straight. We're making an oval now, so we want to make it a bit longer! DoubleClick on "Length" in the right top window, and in the dialog box, the length is 862763. Change this to 4862763 just for fun. Your "Bottom straight" is now longer than the top one.

Now we add two more sections like the other turns we did: Add after, change section type....and we want to make sure this turn section is exactly like the last turn we made...So we first change the "Curvature" of our last section (Section 4) to the same as section 2 or 3 by copying the Radius....



(Never mind the Norwegian text) by using copy (from section 2 or 3) and paste into section 4. You have the curve looking familiar?

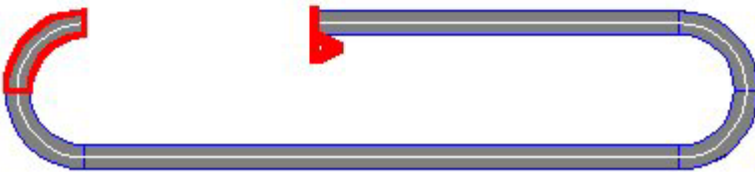


Now, select section 4, RightClick, and "Add After".
 Yoohoo! We're now heading back toward Start/Finish!

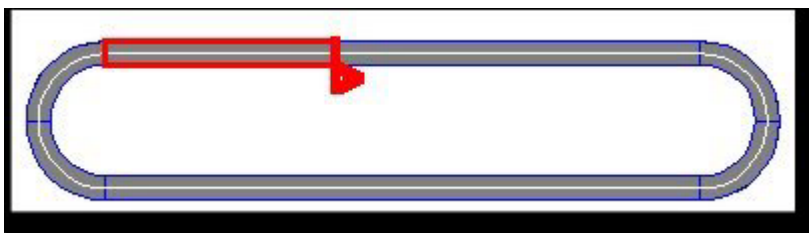
Right Click Section 5, Add After, Change Section Type, you now have a straight bit heading for the GOAL! Only problem is that there is a nasty gap here. And it is vital that you match your last section up to the first section as perfect as possible. If not, Freezes, rapid rides back to the desktop etc is the result. But since we've played it safe here, we've already got the heading right, we just need to fill that gap! And this is easy! Click on section 0, and look under "Longitudinal Gap". You'll find the number 1015748. What?

Here comes the easy part! Fire up the Windows Calculator. Now find the length of your last section (6) which is 862763, add the gap which was 1015748. The sum is 1878511.

Select Section 6, DoubleClick "Length", and paste in the Sum from the calculator - Click OK.



Cool eh? A Perfect track! No gaps or mismatches! **SAVE! SAVE!**



A hint: Try to match up the final sections after you have made a more difficult track...You may now sit back and relax, loosen up your mouse-wrist and neck. We've just started!

The beginners 'guide' to making a GPL track PART 3

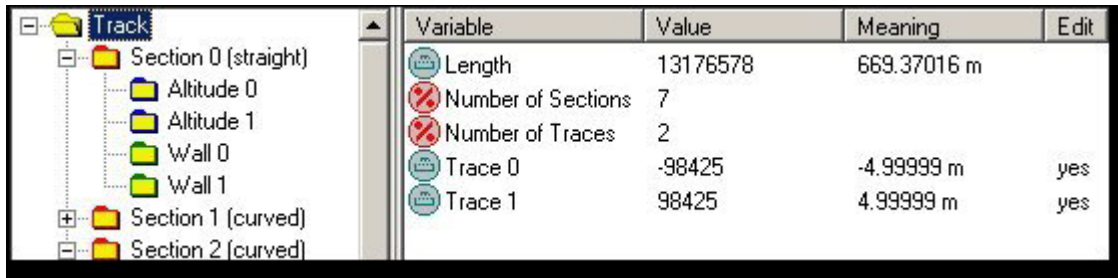
Part 3 Traces & Walls

Ok, we've now got the track shaped like a rather dull oval. But we need a bit more before we can try to compile this. Take a sip of your coffee while you read about...

Traces

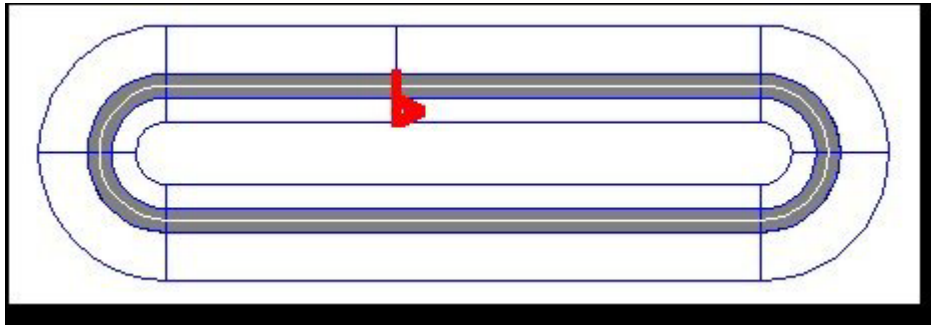
Traces are the blue lines you see on each side of your track now. Their primary use is for defining elevations, but nothing (except dedicated objects) will be drawn outside the traces. And you'll need to keep them clean, when you have added several traces, it can look quite a mess in tight corners, and be difficult to handle. So lets add a trace on the outside of the track, defining the leftmost border.

Start out by clicking on the top of the tree...(TRACK)



Variable	Value	Meaning	Edit
Length	13176578	669.37016 m	
Number of Sections	7		
Number of Traces	2		
Trace 0	-98425	-4.99999 m	yes
Trace 1	98425	4.99999 m	yes

Here you see two traces. Between them is the track, or asphalt. The centerline of the track is defined as zero- negative is right, positive left (this is common in more files, which we'll see later. So remember it!). Right Click on Trace0, and select "Add before". You now have a blue line on the infield, and a New Trace0 at -295275 by default. Right Click on Trace1, and "Add After". Bing! Another trace on the other side of the track at 295275, and read "Meaning": It's 14.99997 metres from the centre of the track. Knowing the track is 10m wide, this gives you a run off of 10m. Not much, so lets move it a bit out! DoubleClick on Trace 3 to bring up the DialogBox, and change from 295275 to, say, 495275. More comfy now with 20m of run off to the outside. Here's how mytrack.gtk looks:



Note: Don't place a trace at position 0. trk23dow will not like it !

Ok, more about the usage of traces later, you want to know that you can have up to 16 traces in total at a track. They need to be planned before adding, but more on that later! Now, this track is still useless, unless we add something beside the track itself, and that means....

Walls

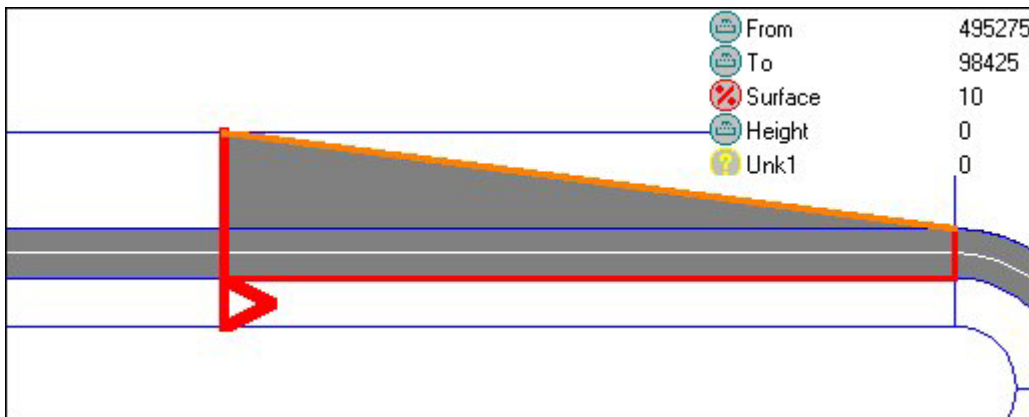
It may sound silly. Flat walls. Doh. But the gray you see on your track is actually a wall too. So, if asphalt is a wall, so is everything else. Here's a short list of wall (surface) types:

Surface type	Is.....	And behaves like....
1	Tarmac (Asphalt)	Well, er, like a dry racing track.
2	Concrete	A bit more slippery than tarmac
3	Curbs	Same grip as Concrete, or therabouts
4	Grass	Slippery!, bumpy
5	Dirt	Dusty, slippery, bumpy

6	Gravel	Very Dusty, sticky, slippery, bumpy
7	Banking	Sticky, soft, dusty
<i>Those were the most common types, but then you have these walls for other purposes....</i>		
2048	It's there!	You can drive straight through it, nice for bushes
2055	Soft and sticky	Great for walls of haybales or tyrewalls
2056	Hard! Devastating!	Excellent for armco, rows of trees, fences..
10	Um...	These need to be on the extreme left, otherwise your track won't load!

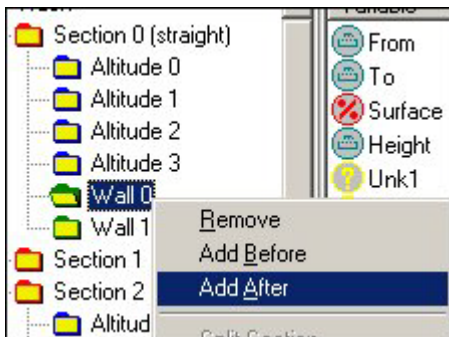
Ok, so how do we go about adding these walls? Thanks to Peter, there is a great feature in GPLtrk that will save us loads of work, so we'll add the basic walls first, starting with Section 0. The plan is as follows: we'll need grass on each side of the track, some fencing on the right, and some trees on the left to block the view into infinity, and the wall type 10 to make our track load. But in addition to this we'll have to start the mytrack.tex to tell trk23dow (the compiler) what we want to be drawn on a 2056 wall! Let's do the walls first...

Start by zooming in a bit on Section 0. Double click the gray track itself, so you have an orange border indicating this is the wall that you're reading about in the top window. On the tree in the left top window, you'll see one of the small folders open; this indicates which wall you are working on. When you're getting a lot of the walls (10-18 walls is normal), this is nice! Click on this open folder called Wall 1, and you'll see that this is type 10, the leftmost and required border. You'll want this border to be on the outmost trace, which we know is at 495275. And this wall is at 98425 isn't it? Well, DoubleClick (aaargh, from now on it's DC, ok? And NOT Squarehead!) "FROM", and paste or type 495275 to replace 98425. Now it looks a mighty mess; like this to be exact:



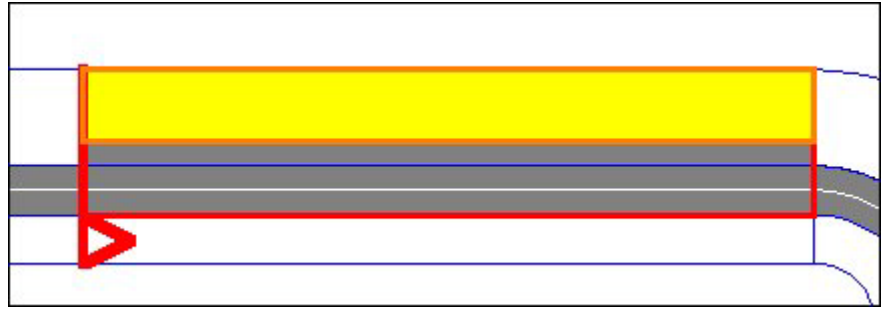
Um, we'd better bring the "TO" Value to the same place (Get it? "From" = The start of the section, "To" = The end!)

Now, this still isn't what we're after. Since we can't get a texture on a type 10 wall, we'll need something to disguise it. A 2056 type wall is very nice for trees! What you can see now by selecting Wall 0, is that it stretches from the right side of the track all the way to the left border. So we'll need something in between. Right Click in the tree, and select "Add After".

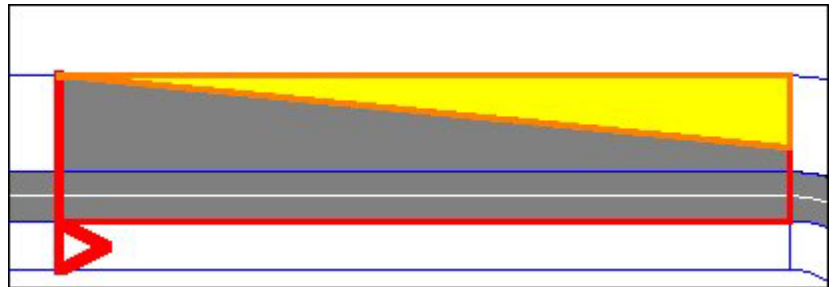


Even though it's not too easy to see you now have a new wall. By default this is surface type 1 (asphalt). We want to change this to a 2056, so we click on the new wall (which is now wall 1), click on surface type, and in the box you enter 2056. Now you should have a bright yellow wall, and it should all look like the pic below.

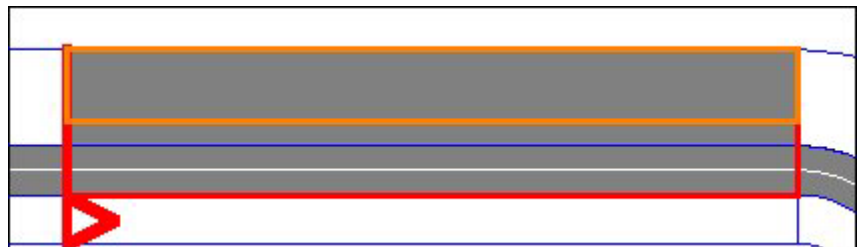
The thing is, your 2056 wall is now 15m wide. No good, you want to tuck it right up to the outside (wall 2, type 10). But a wall *must* have some width, or the compile will fail or go badly wrong. So we take the positioning of the outside wall which is 495275, pull off a few digits to give it width, say, 495273 and add this value to the From and To fields of the 2056 wall.



Here, the from value is set to 495273...and when the To value is changed as well, the yellow wall is so thin it won't be seen. But now the track stretches all the way out again, so we need something in between....Again, right click in the tree, chose Add After...



The orange frame now shows the new wall 1, DC this wall, and change surface type to 4....



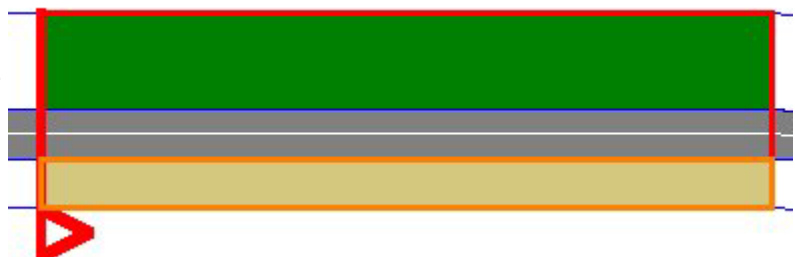
Bingo! Grass! Now we want that grass to tuck right up to the edge of the track; so we have to change the To and From values again. Now, we know that the left trace at the edge of the track is at 98425, so this is where we want the grass in mytrack. What you staring at? Change the To and From, from 198424 to 98425.



Now we need to add two walls before Wall 0 to add something to the infield. Just RC (RightClick) on Wall0, Add Before, RC, Add Before. Change the surface type of Wall 0 to 2056 (so we can't drive off limits!) and this is how it all looks now:



Now, that Wall 0 needs to be moved a bit so it doesn't cross that Trace at -295275, so change it's To and From to -295275. To get some grass on the right as well, change surface type of Wall 1 to...hmm, 6 (Gravel), and it should all look like this:

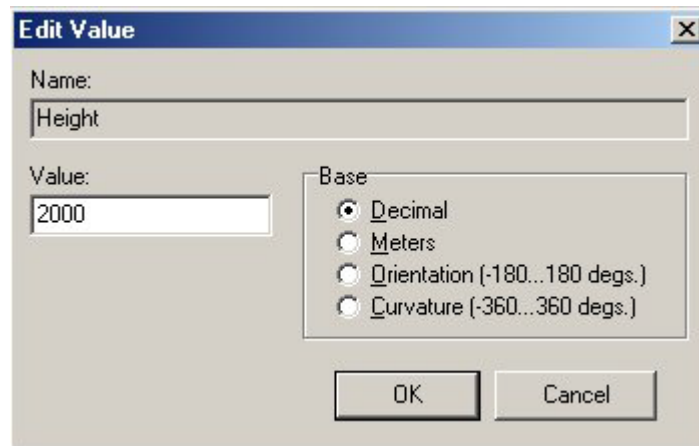


But Hey! Look at the width of our wall 0! It's 0! Bad!

You adjust this by moving the wall next to it a bit, this wall 0 is From and To -295275, so let's change the values for Wall 1 to -295273. Wall 0 now has the width of 0.1016mm! But it's something! Enough even!

Track	Variable	Value
Section 0 (straight)	From	-295275
Altitude 0	To	-295275
Altitude 1	Surface	2056
Altitude 2	Height	0
Altitude 3	Unk1	0
Wall 0	Unk2	0
Wall 1	Unk3	0
Wall 2	Unk4	0
Wall 3	Width@Start	0
Wall 4	Width@End	0
Wall 5		

Ok, time to breathe. For these walls to be effective retainers, they'll need height. On the right, we plan to use armco, which is normally about 1m tall, so go to Wall 0, DC on "Height", and type 2000 (=1,016m). Move on to the outside 2056 wall (wall 4, not 5!), and a typical tree height is um...say about 5m. So enter 100000. Wall 5 needs height too (the outside boundary wall must have height too!), so add a token height to this wall of 2000.

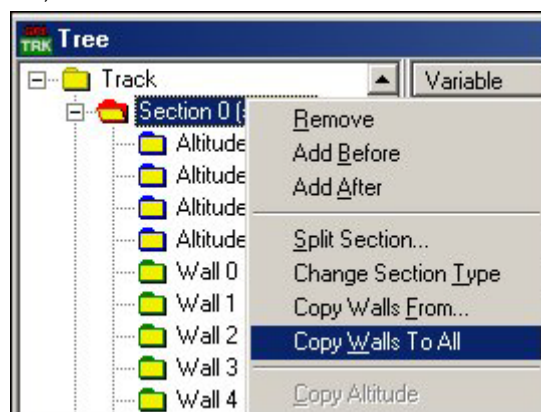


See something else? Right, wall 5 doesn't have width...so move it To and From 495277 (it used to be 495275). Whew! Worst part of walls teaching done! Now SAVE!

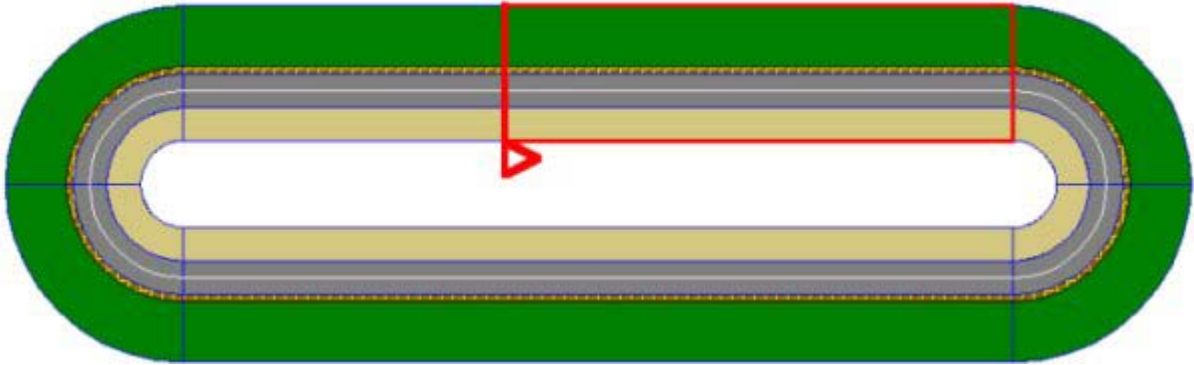
Right now, the only texture you'll see on those trees is the error texture, because we haven't defined them yet. Ready for that? Let's add one more wall!

Click on wall 3 (the grass) and select Add after. You'll now get an asphalt wall behind the grass. Change this wall (by clicking on it) and change surface type to 4 (grass), move it To and From 136849. Now change surface type of the grass wall next to the track (wall 3) to 2055, give it a height of 10000. This will be a wall of haybales, more explained later! Now, DC on "Unk 4" while you have selected the wall 3. Enter 98, click OK. More on this later...muahaha!

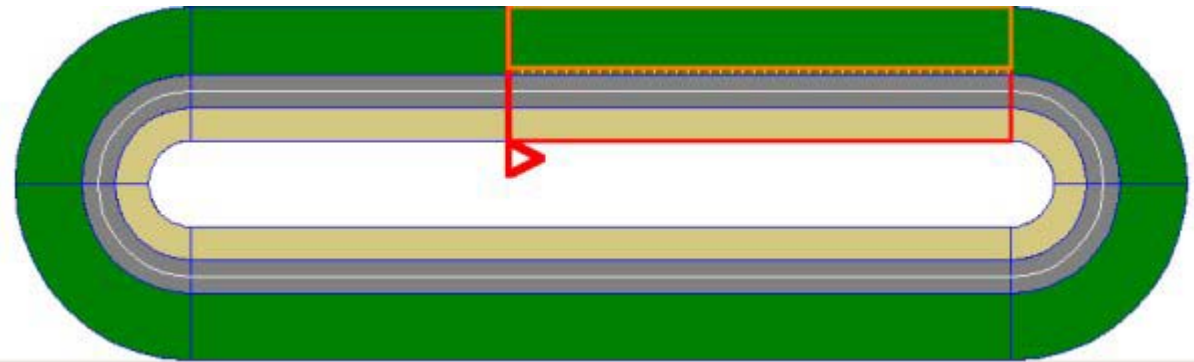
Now go to the tree on the left, select/highlight Section 0 (be sure you have Section 0 - the section with all your nice Walls - selected !!!) Right Click, and select "Copy Walls to All".



Now this is how your track should look! SAVE!



The thing is, you now have a wall next to the track all around it: boring! Double Click on one of these walls, say wall 3 section 6, Right Click in the tree on wall 3 while this section is highlighted, and choose "Remove". ZAP! Wall gone! Change The From and To values of the grass to bring it back to the track, change to 98425. Repeat this for all sections except Section 0, we want to keep the hay here! Now it looks like this:



This is how it looks now! All well? Then SAVE, and move on to how we are going to make it all look beautiful!

The beginners 'guide' to making a GPL track PART 4

Part 4: Adding Textures!

Ok, so now we got our GTK done for the moment. Now we need to add textures. The textures are defined in a text file called mytrack.tex. The file defines the name of the texture, how it is stretched/mapped, and if it has height/width, which textures to use on the top, bottom, sides, front and rear. Lets look at that! Start notepad, and open the mytrack.tex you made, remove the contents, and start by adding these lines:

```
0 0 0 10.0 10.0 error
1 0 0 10.0 10.0 asphalt
4 0 0 10.0 10.0 grass
6 0 0 10.0 10.0 gravel
2055 0 0 10.0 10.0 hay
2055 0 1 10.0 10.0 hay
2055 0 2 10.0 10.0 hay
2055 0 4 1.0 1.0 hay
2055 0 5 1.0 1.0 hay
2056 0 1 10.0 10.0 trees
2056 1 1 10.0 10.0 barrier
```

So what does this mean? We're defining textures. Remember surface type 1 = tarmac (asphalt)? Let's look at them again, and specifically the 2055 wall (the hay) to explain things.

First: the names here will be needed as mip-files in the track dir in GPL (not necessarily the dir you are making the track in). These files are included in your zip.

Ok, let's look!

```

wall type          stretch sideways & upwards
2055 0 0 10.0 10.0 hay
2055 0 1 10.0 10.0 hay
2055 0 2 10.0 10.0 hay
2055 0 4 1.0 1.0 hay
2055 0 5 1.0 1.0 hay
Unk3 value        name of texture
                  which surface

```

Line 1 says;

Wall type = 2055, Unk3 = 0, side = 0 (floor) Stretch 10m sideways, 10m upwards, use a Mip called hay.

Easy!

I've explained wall types/surface types. What is Unk3?? It's Unknown3 which is known; it's a sub value. Look at the lines above: there's one line that starts with

2056 0 - this means, use texture 2056 - 0. Which means trees.

2056 1 - means, use texture 2056 -1, which means barrier.

Basically, the same type of wall, but the sub-value defines the texture used. The Unk3 value can be set in GPLtrk, and in fact, let's go back to mytrack and fin it so that the inside wall uses 2056 Unk3 = 1 (which is the barrier).

Here I'm fixing Wall 0 of section 0. Please do the same for wall 0 of all the sections, and SAVE.

Variable	Value	Meaning	Edit	Comment
From	-295275	-14.99997 m	yes	
To	-295275	-14.99997 m	yes	
Surface	2056	2056	yes	
Height	20000	1.016 m	yes	
Unk1	0		yes	
Unk2	0			
Unk3	0			
Unk4	0			
Width@Start	2			
Width@End	2			

Edit Value

Name: Unk3

Value:

Base

Decimal

Meters

Orientation (-180...180 degs.)

Curvature (-360...360 degs.)

Ok, now we got the Unk 3. This means that you can have a LOT of walls which are type 2056, 4, 2048, whatever, and none look the same, because you define a different texture in

the Unk3 (subtype).

Even more Good news, look at 2055 again, the lines all look the same, except for the 3rd value from the right. This defines which side gets which texture. 0= flat, the ground. So, obviously, grass, gravel and Asphalt only needs the ground defined, while the 2055 needs the other sides as well, since it has got width & height. Here they are:

0= Ground

1= Sides (left and right)

2= Top

3= Advanced ! See trk23dow guide

4 = Start of wall

5 = End of Wall

6= Advanced ! See trk23dow guide

You can, if you like, use a separate texture for each side of walls with height.

Got that sorted? Good! The first line, the error.mip, is in case you missed something, or did a typo in GPLtrk. The error.mip is usually a very bright and easily seen one, so you can see where you've made a fool of yourself!

OK! Now let's just add a couple of objects, and by cheating (avoiding the more advanced stuff) we can compile and take mytrack for a drive!

The beginners 'guide' to making a GPL track PART 5

5. Adding objects - now we're speeding!

We've got out tex sorted, and lucky you - I supplied the necessary textures (mips) you need. You have the other files ready? Let's add an object!

Open the file you saved as "mytrack.tso". Tso = trackside objects. Now write this...:

```
# Trackside 'Sets': 1
Set List
[ 0]: 3, 51.0, 15.0, 0.0, 90.0, 0.0, 0.0, 1.0, grand
0InfieldObject
0OutfieldObject
```

Save the file!!

What we're doing here is to tell trk23dow (the compiler program) to add an object next to the track, the object is called "grand" (which is a 3D object included in the zip for your convenience). What we're saying (this is simplifying it a LOT, but this is a newbie tuto, so be happy!) is:

Trackside 'Sets':1 = the numbers of objects in total added to the track

Set List = here it comes Mr.program

[0]: 3, = Object number 1 (is 0 in computer terms!), 3 means it's an object we intend to tell you the location of in normal terms, which are:

51.0 = Distance from the startline

15.0 = Distance from the centre of the track (positive = left, negative = right)

0,0 = It's on the ground! 12.0 would have put it 12 metres up in the sky.

90.0 = Rotate the object 90 degrees around the centre axis horizontally (this is decided by the object, and you'll do a trial and error job here in the future!)

0.0 = Don't tilt it (Rotate the other way, vertically)

0.0 = Er..

1.0 = Er...

Got that? Good!

Now if you want another object, just add a new line, which would make it look like this:

```
# Trackside 'Sets': 2
Set List
[ 0]: 3, 51.0, 15.0, 0.0, 90.0, 0.0, 0.0, 1.0, grand
[ 1]: 3, 51.0, 15.0, 0.0, 90.0, 0.0, 0.0, 1.0, anotherobject
0InfieldObject
0OutfieldObject
```

As for the Infield & Outfield objects, read about it somewhere else on this web site when you feel ready! A few points to note:

- *Don't* position an object across traces or walls: It will clip (muuuahahahhaa!!). This means it will vanish/blow up/ go mad when viewed from different angles. You'll discover clipping in time!

- You may add a comment to the end of each line; a single word, don't use numbers!
example:

```
[ 0]: 3, 51.0, 15.0, 0.0, 90.0, 0.0, 0.0, 1.0, grand turnone
```

This is very handy when you get into adding objects. A clue: Crystal Palace has around 550 trackside objects.

- If you think a bit, working in Excel, and copying/pasting into notepad can be really helpful!

- Flaggers, gridboxes, trees, wah wah has to be added. You have an idea now what can be made from walls.

Ok, now let's do a fair bit of cheating. Do the following:

Copy **mytrack.set**, **mytrack.grv**, **mytrack.fb** from the attached files, and drop them in your /mytrack dir, overwriting your old files. To understand these files we're leeching, you'll need to read up [trk23dow!](#)

Open GPLtrk, and chose **File-> Export Track File**.

You now export the trk file from GPLtrk. 'mytrack.gtk' is yours only, a file editable, while a trk can *not* be edited. **To avoid cheating**. So don't share your GTK! And you'll have to mod the GTK, export trk every time you change something.

Make sure you save it as "**mytrack.trk**". Now Open a Dos window with the path of the working dir (to do this, have your working folder open in explorer, and chose Start->run->command. Voila!)

If you followed this tuto, the path should be:

C:\GPLedit\mytrack

Then type the following in the DOS window:

```
trk23dow -t mytrack
```

Hit Enter...and BLITZ! It finishes without drama (this process will take much longer for a proper track, depending on CPU power).

Please note: trk23dow uses mytrack.GTK, while GPL uses mytrack.TRK. They must be the same track for things to make sense !

Now, go to your mytrack dir, there should be a file called **mytrack.N3D**

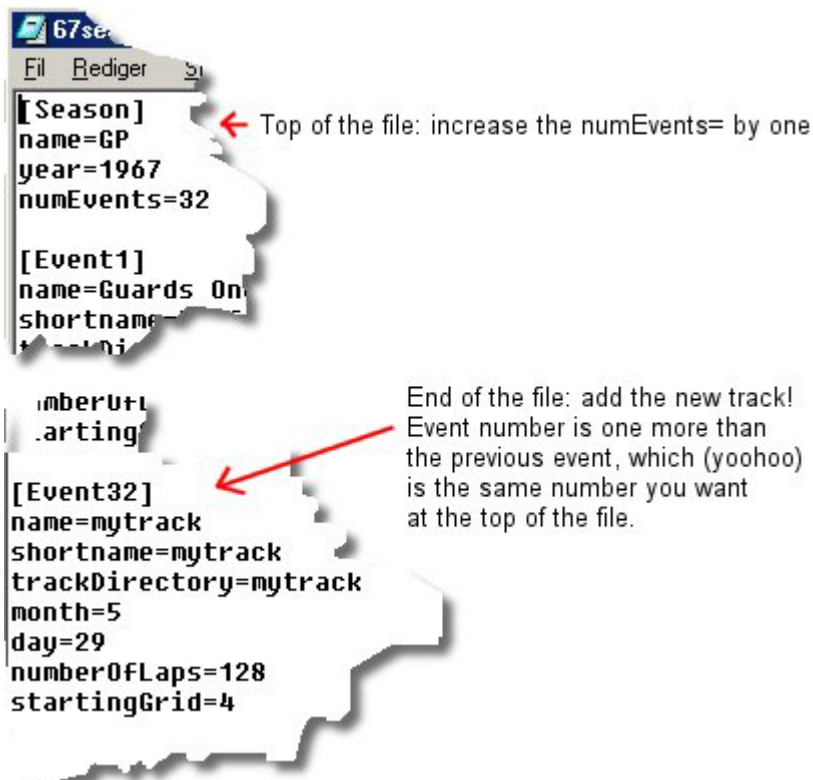
Rename it to 'mytrack.3do'

Copy the entire folder, and paste it in underGPL/tracks/

You now have a new track! For it to work, you'll need to borrow a few files from one of the other tracks, say Silverstone. Copy the following files from silver to mytrack:

silver.dat
track.ini
Pass1.lp
Pass2.lp
race.lp
minrace.lp
maxrace.lp
pit.lp

Rename 'silver.dat' to 'mytrack.dat'. You're done here....now go toGPL/seasons/ and find '67season.ini' and 'gp.ini'. These will have to be modded slightly for the track to appear (see below)



Ok? You ready? This is where I can't be bothered to write anymore. You're on your own! If you get a 'could not load track' error, files are missing!

Oh, and: when you zap through the programme covers in GPL, mytrack will show up as Silverstone. And you'll need to SHIFT-R as soon as you hit the track; you'll find out why! :o)

Have fun hotlapping the terribly dull oval!

Creating a TRK

GPLTrk

GPLTrk is the program used to create TRK files. It has been designed to work with GTK files, so you cannot load TRK files, only export a TRK for use with trk23dow/GPL.

Your GTK/TRK consists of 4 sections:

1) Traces

These are lines going all the way round the track, parallel to the centre-line. The Altitude curves follow these lines.

Traces

Curved Parallels

Traces are a bunch of lines, parallel to the centre-line of each section that define the positions where the altitudes are defined. You can have between 2 and 16 traces on a track. Parallel may sound a strange word to use, but because the track is essentially a 2 dimensional construction (longitude and latitude), you can pretend that corner sections are straight - then everything makes sense !

Choosing Traces

A typical track will have traces defined where the interesting elevations are - the edges of the track, where kerbs are, where spectator banking exists etc. When designing your track, you should make sure that the traces are well-used - a pair of traces for the extremities of the track are normal; a pair of traces for one piece of banking for one short trk section is not acceptable.

Never use a trace near the track centre (latitude = 0). trk23dow will not like you, you run the risk of getting all sorts of weirdness (mostly to do with the racing groove), and it's really not necessary to have one there.

It is much better to have one trace for the edge of the track, one for where the kerbs are, a couple for slopes and an extremity trace. That's 5 per side, or 10 total, which is a good number.

If you don't bother with the unk4 flags, you will find that as you add more traces, your <trackname>.3DO file will get much larger, and the FPS will decrease due to the number of polygons created by all those traces.

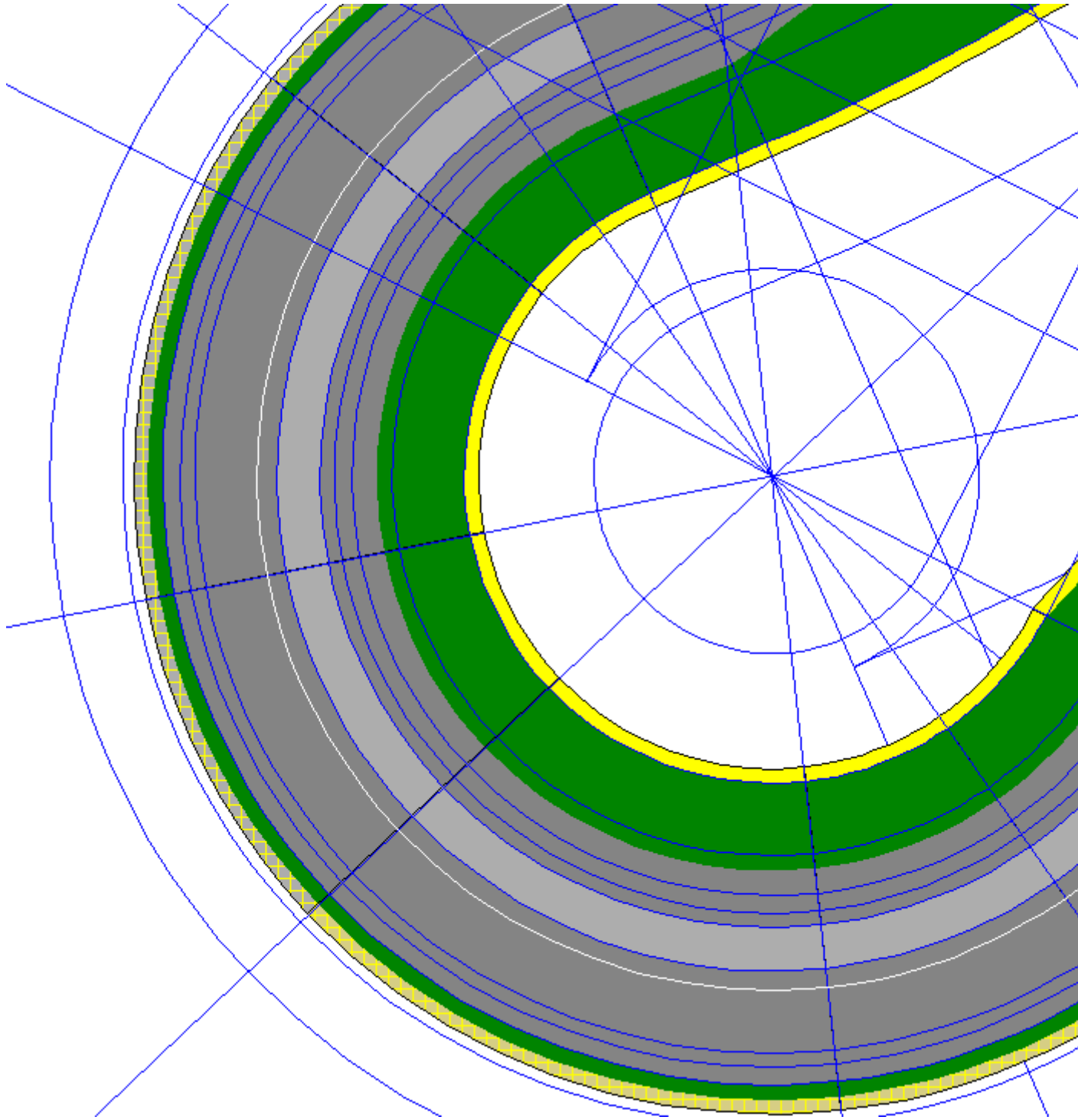
Here are two examples from the Nurburgring:



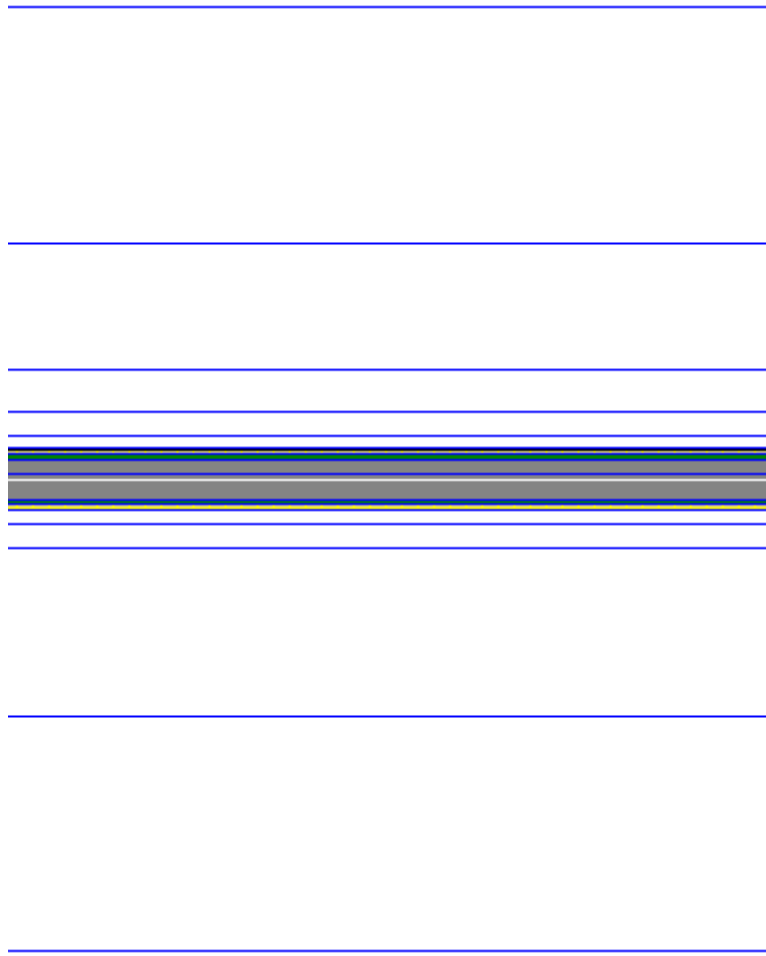
Right hand trace shown above is the same as the middle trace below - good re-use of traces by Papyrus!



See how noticeable the location of the traces above is. Now look at the TRK below (showing the second screenshot) and spot which traces are which:



Here is a pic showing exactly where all the 16 traces for the Nurburgring are located. See how there are a few used for the extremities, far from the driving line, and more where most of the action takes place (the run-off !). One trace is used near the middle of the track for the Karussell etc.



2) Sections

Your track will be constructed from sections - a combination of straight and curved pieces (the curved pieces are arcs of a circle).

Sections

Straights and Corners

Your entire track is constructed from Sections - either **straight lines** or **segments of a circle**. Every section follows on from the previous one until you get back to the start of the track again (no hill climbs or drag strips :-)

Sections form the basic structure of the track - the altitudes are 'parallel' to the centreline defined by the sections, while the walls create the actual driving surface in each section. Creating sections in GPLTrk is easy, but you will soon run into problems when changing the layout of your track, and when you come to aligning the first and last sections.

Creating Sections

Start GPLTrk and create a New track (File->New). You will be given a single straight section - the grey block. Notice the red arrow - showing you that the cars will head in that direction (left, right, north, hubwards - absolute direction is irrelevant ! All that matters is the overall shape of the track).

Ok, so we have a 50 metre long piece of straight (click on the grey block to get information in the top window):

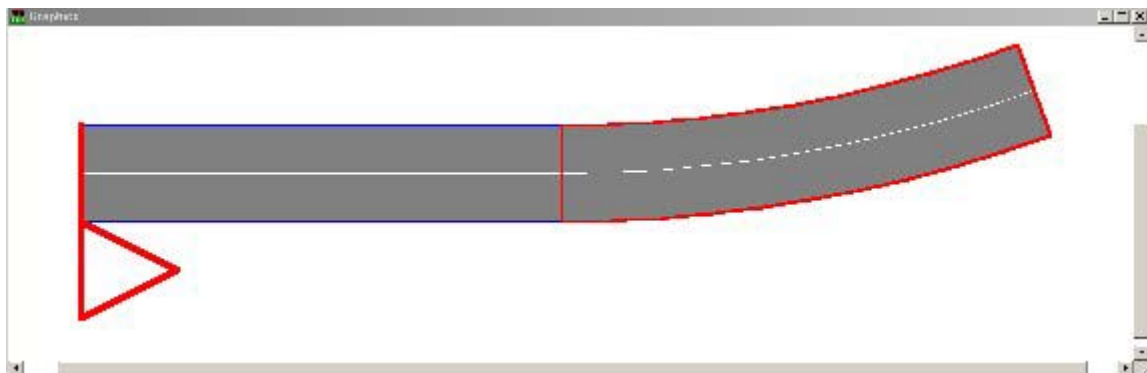
Variable	Value	Meaning	Edit	Comment
Type	1	straight		
Start	0			
Length	984252	50.000002 m	yes	
Orientation	0	0 °	yes	
Orientation mismatch	0	0 °	yes	
Longitudinal gap	-984252	-50.000002 m	yes	
Lateral gap	0		yes	

[Section 0 - a 50 metre long straight]

Wow - what a track. Time to add another section. On the tree window (left in the pic above), right-click on Section 0 (straight), and choose Add After. You will be given a duplicate of that section - another 50m straight.

Corner->Straight->Corner->Straight->...

So far, you have 2 straight sections. Boring. Right-click on Section 1 (straight) in the tree view, and choose Change Section Type. Voila - a corner section, turning 20 degrees to the left:



[50m straight, 20deg corner]

Section Length

Look at the first pic - by double-clicking on the Length word, you will be given a dialog box to change the length of the straight section. This dialog box is used for all values in GPLTrk- it is recommended that you use the Decimal setting in preference to the Metres setting (greater accuracy). For corners, use Orientation for the starting angle, and Curvature for the angle change (both in degrees, where a positive value turns the section to the left, negative to the right). 1 metre = 19685.03937 in the 'Decimal' setting.

Edit Value ✕

Name:

Value:

Base:

- Decimal
- Metres
- Orientation (-180...180 degs.)
- Curvature (-360...360 degs.)

[Edit Value dialog box]

Change it to 1984252. Oops - the straight now overlaps the corner. We need to fix that, or weirdness will happen in GPL... See the Gaps section to fix the problem.

Corner Radius

Simple enough - this is the radius of the circle, that this corner is part of. Let's take a 10metre width track as an example- 5 metres will be either side of the centreline, where the radius is measured. Thus the radius at the apex will be 5metres less than the section radius, and the outer radius 5metres more. For most corners this is not a problem, but for hairpins, you must make sure that the section radius is sufficient to allow for the apex radius !

Also, very tight corners in GPL cause bad FPS - this is something to do with the distance between visibility boxes, which get squished the closer you get to radius=0.

Orientation

Usually, you will want to leave this alone, unless you have Orientation Mismatches (see later), or you are changing the direction of the first section. Value is in degrees, where 0deg is pointing ->, 90deg ^, 180deg <- and 270deg v.

Curvature

The curvature of a corner is the direction change, in degrees. A hairpin might change by 180degrees. BUT ! you must not create corners with curvatures of more than about 120degrees, because trk23dow will throw a wobbly, and your track will be quite weird to drive (invisible bits, unexpected physics, etc.). Positive curvature makes the section turn anti-clockwise (to the left as you drive), negative is clockwise (to the right as you drive).

Gaps

For the track to be compiled correctly, and to *not* cause GPL to lock up or crash when you drive around, you must make sure that every section is aligned correctly with its neighbours. Taking the straight/corner problem we have created, do the following:

Select the corner section - section 1:

Variable	Value	Meaning	Edit
Type	2	curved	
Start	1984252	100.8 m	
Length	984252	50.000002 m	yes
Orientation	0	0 °	yes
CurveCenterX	984252	50.000002 m	
CurveCenterY	2819674	143.23944 m	
Curvature	119304647	20 °	yes
Radius	2819674	143.23944 m	yes
Orientation mismatch	0	0 °	yes
Longitudinal gap	-1000000	-50.8 m	yes
Lateral gap	0		yes

[Gaps]

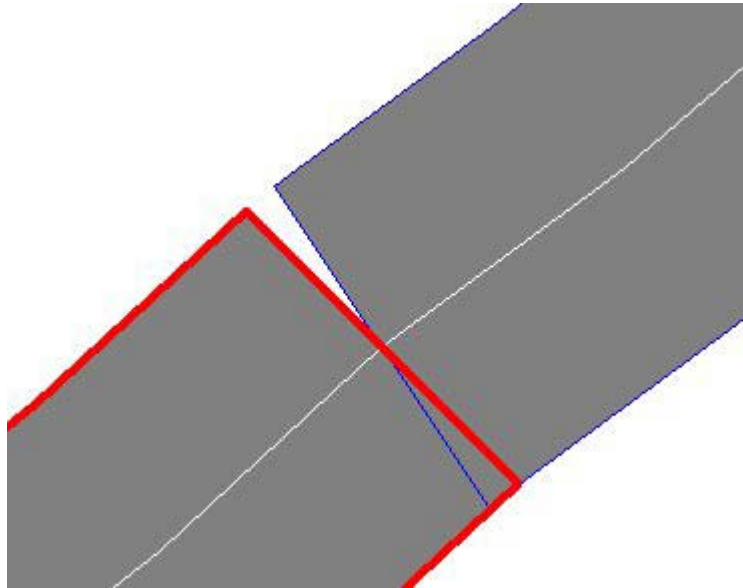
Look at the Longitudinal gap - it is -1000000, or -50.8metres. Double-click on the word LongitudinalGap, and change the value to 0 decimal. This will reposition section 1 so there is no gap (/overlap).

More Gaps

There are three types of gaps/overlaps you will come across - Orientation Mismatch, Longitudinal Gap and Lateral Gap.

Note that section overlaps can sometimes be spotted because they cause spike in force-feedback steering wheels.

Orientation Mismatch - this means that this section does not have the correct Orientation value, when the previous sections' Orientation+Curvature values are compared. Look at this diagram - the sections clearly have a gap. Usually, this will be happening on a very small scale, but this is plenty to crash GPL, so make sure that all Orientation Mismatches are 0 !



[Orientation Mismatch = GPL crash]

Orientation Mismatches can be fixed in one of two ways:

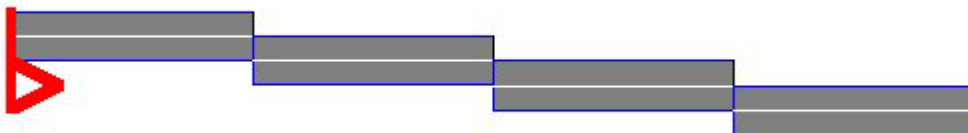
- 1) If the preceding section is a corner, you can adjust its Curvature so that it fills the gap. Remember that the scale of Curvature is half that of Orientation, so if you have an Orientation gap of 1000, you will only want to change the Curvature by 500.
- 2) Set the Orientation Mismatch to zero. This will require you to carry the mismatch on to the next section, so 1) is preferable.

Longitudinal Gap - means that the sections have either a gap (value is >0) or that they overlap (<0). It is ok to leave an overlap if it is small (500 or so), but all gaps must be corrected. 1) If you have an Orientation Mismatch, FIX THAT FIRST !

- 2) If the preceding section is a straight, simply alter the Length of that section to fill the gap/reduce the overlap.
- 3) If the preceding section is a corner, you can alter the Radius of that section. This will also affect the Lateral Gap, so use carefully to avoid making a bad thing worse.
- 4) Set the Longitudinal Gap of this section to 0. Again, this will carry the Gap on to the next section.

Lateral Gap - a sideways mismatch. These gaps cause the fewest problems, but for a decent track you won't want any, so:

- 1) Fix Orientation Mismatches and Longitudinal Gaps FIRST !
- 2) Find a preceding corner section, and adjust the radius. This will also change the Longitudinal gap, so take care.
- 3) Set the Lateral Gap to 0. If you have several pieces of straight, you can spready the Lateral gap out over all those sections. Here is an exaggerated example:



[Stepped Lateral Gaps]

That Last Section

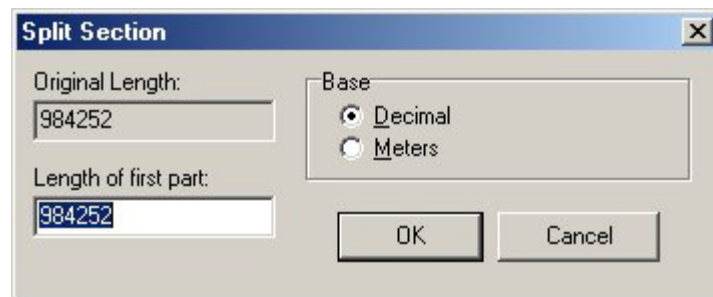
So, you've created a 40km track, and the last section doesn't match up. To fix this, follow the guidelines listed in Gaps. You might need to fix several consecutive sections (starting with the furthest

away from the finish!), because you have:

- 1) A straight as the last section - not much you can fiddle with here.
- 2) A 'good' piece of track leading up to the final corner - i.e. you have known radii of the corners that can't be changed. Here you'll have to change the length of sections/curvature/radii etc. of sections that have significant effect on the end of the track. By this, I mean that changing the length of a section before a hairpin will be like changing the length of a section after the hairpin, because they are mostly-parallel.

Splitting Sections

You have a long straight section and you really really want to have an escape road, or a different set of walls for half of it. Don't rebuild the track ! All you need to do is right-click on the section in GPLTrk's tree view and choose Split Section:



Give the length of the first 'half', and GPLTrk will split the section into two. However ! If you have altitudes, GPLTrk will not retain the structure that *you* want - it will only make sure there are no altitude gaps, so you will probably end up with a sharp angular piece of track.

Section Lengths

Don't create too many sections for your track, and make sure there are not a large number of short sections (20 metres or less). As an example, Goodwood is 99 sections over 3.8km, which is quite a lot. Solitude is 175 sections and 11.4km which is about right. The original Papyrus tracks range from 26.9metres-per-section for the Nurburgring to 87.1metres-per-section for Monza. The 'Ring is a bit of an exception, because the scenery is relatively dull, with few buildings, people etc.. For your tracks you should be aiming to be in the 50metre-per-section range, but always avoid lots of short sections, especially tight corners !

3) Walls

Walls is the term used to describe the way each Section is broken down. These walls are all the flat regions of the track: asphalt, grass, banking, concrete; as well as all the real walls: armco, hedge, rows of people, etc.

Walls

Walls is the term used to describe the way each Section is broken down. These walls are all the flat regions of the track: asphalt, grass, banking, concrete; as well as all the real walls: armco, hedge, rows of people, etc. The way the wall 'acts' depends on the surface type it has, so let's look at the surfaces....

- 1- Asphalt
- 2- Concrete
- 3- Curbs
- 4- Grass
- 5- Dirt (makes dust!)
- 6- Gravel (makes dust!)
- 7- Banking (makes dust!)
- 8- Some sort of very vicious surface, probably was intended to be water.

10- Left-most wall
2048+ - a wall that you can crash into - e.g. armco, hedge, brick wall etc.
2048 - Drive through (grass verges etc)
2049 - Very hard, armco, etc.
2050 - Same as 2049
2051 - Soft, 'elastic'
2052 - Like 2051, but not as 'elastic'
2053 - Soft, hay bales
2054 - Soft, car sinks into (hedge?)
2055 - Soft, softer than 2054
2056 - Hard, not metal, cars bounces off a bit
2057 - Hard, metal, armco
2058 - Soft - hay bales
2059 - Hard, metal, armco
2060 - Bouncy rubber
2061+ - Drive through

Ok, when you start out making a new track (gtk-file that is), you have by default 2 walls, one that is surface type 10, and one that is surface 1 (Asphalt). The type 10 needs to be the leftmost wall, it defines the outer boundary of the track, so no matter how many walls you add, the 10 needs to be your leftmost wall.

The walls are manipulated in GPLtrk, you start out with the asphalt and the 10; you'll at least need 2 more walls to define the driveable area, and they both need height. To stop the cars from toppling into neverland, you need to use surface 2056 or higher.

Most likely, you'll be adding lotsa walls per section, and you'll have one hell of a time reducing this number when release day comes to get nice FPS....

How to add walls: click on one of the existing walls in GPLtrk, rightclick and choose 'add after' or 'add before'. Then you'll need to wibble with the width until you're right, BUT: a wall **must** have width! It need'nt be much, but compilation will fail if you don't do this. Some walls are required to be very thin; for example fences, or rows of people. Make sure you give them width!

If possible, avoid having your walls crossing traces. Weirdness will definitely occur, especially if you are planning on adding altitudes.

Height

You set height for the walls by selecting the targeted wall in GPLtrk. There hardly is any limit to height, but there are a few things to consider. All walls with height will have to be defined with textures for all sides in the .tex. (see section on textures).

Remember that if you add height to a wall, a wall coming directly behind it will need to be even higher, and it'll need another wall behind it to fill the void. If you add walls as earthbanks, barriers or simlair, also remember that any 3dos placed on top of them will have to be adjusted in height in the tso.

You can also in fact, give the walls negative height. This isn't used very often, only in cases where there is a rapidly rising tight hairpin and so on to 'camouflage' the inevitable gap on the inside of the corner.

When working with height: if you have a wide wall, and it sits on top of a trace, a 20++ wall will usually not 'bend' with the terrain, with ugliness as the result. A trick is to use a different surface, 4 or whatnot.

The Unknown values, and their known uses..

Unk3= chooses texture in conjunction with surface type.

Unk1 has a snazzy feature: in the case where you use a wall with height and thickness as armco, barrier, whatever, and you have a repeating texture being chopped off ('ants' when it was supposed to show 'pants'), you can add a value here to 'push' the texture to the right. The value is the same as elsewhere (20000 = roughly 1 metre). Experiment, this is very useful!

Unk4= The big business.

Whenever you have walls with height and width, you'll need polygons on the ends, else you'll have ugly green gaps. The fences & rows of people may not need this, because they can be made very thin. To add these polygons, you'll have to do some unk4 work. The exciting thing with unk4 is that you can add several values to it that do different things!

The most used values are:

unk4=2. Used on walls where you want the texture to stretch across only once. Very handy when you make a firewall or similar.

unk4=8. Used where you want one texture to be mapped only once..if there is a certain patch of grass on a wall...the lay-by's at Goodwood is an example.

unk4=32 : adds a polygon to the start of the wall. Remember to define this in the tex!

unk4=64 : adds a polygon to the end of the wall. Remember...wob wob!

unk4=96 : adds polygons to both ends.

Now, you usually want to use these in conjunction with the texture stretching only once, so add 2 to that unk 4 value for every side (34,66,104...).

The easiest way to observe this is by not setting an unk4 value for the asphalt: you'll have odd texture mapping!

Walls with height and texture behave strangely when you make a corner: the textures will stretch on the outside, and cram on the inside. (rows of people will be very fat when in bends!). To remedy: add 2 entries in the tex; one texture for your 2056 wall on straights, and one for corners. This usually is enough.

When using kerbs and special grass, you may want the textures to stretch only once across the wall (unk4=2): if you're next to a trace with altitude, this will leave gaps in the track, so you need to plan carefully!

unk4 is also used to control optimizing, for a complete rundown on the unk4 codes, please check the trk23dow/GTK part. But while we're at walls, another neat value is that of 131072. This makes sure a polygon isn't added to the top of a wall with height - always use it on rows of people, fences, treerows, anywhere that you won't see the top! Saves 3do size, and ups performance.

1024 should also be added to any wall that is to the right of a clipping problem: if you have a seriously banked corner, you may experience clipping; this often helps.

There are loads more that you can figure out and add up; again, check the trk23dow/gtk section, but always remember unk4=2,32,64,98,1024 & 131072. These are set on individual walls while the others are set as a flag on wall0 of a section.

In general, try to keep the number of walls down. You can have lots in each section, but performance will suffer. The good thing about using walls where the elevations are constant (like the banks at Goodwood), is that the texture on the farside will not be drawn, and this increases performance. In fact, this is better FPS-wise than using altitudes. The downside is

that if you use a 2056 wall, wide, then want a row of people on top of this made with a wall, that wall needs the height of the 1st wall + that of the people, which again means that the mip will have to be 2x the height, since the lower half will be invisible.

To finish off, let's look at a typical section of track: this one taken from Goodwood:



To the right of the track, I start with a 2056 wall with some height. The texture used is a grass verge. This is an airfield track, so it's the only way to cammo the end of the world!

Next up is a surface 4 wall: slippery grass, non-smoker.

Here we have a surface 7, this is slightly more spooky to drive on, and it will create a cloud of dust when you run on it. Still looks like grass because of tex-work.

Next up are two surface 3 walls: one for the actual kerb, and one for the white line/transition mip. Usually needs unk4=2, but beware of traces!

Asphalt, Must have unk4=2

Surface 3 again: white line/transition (I have a fetish for these)

Some more grass

2056 bank starts (hard bank this!). This bank uses different textures for top & sides. Height is about one metre.

In this bank is another wall that looks like a fence, and another wall being a row of people (the 2 2056's). This is followed by a type4, a 3do, a type6, then the obligatory 2056 w/height (row of trees on this occasion) and finally the type 10 wall.

Whew!

4) Altitudes

Flat tracks are okay, but hilly tracks are more fun. Altitudes are cubic functions defining the height of each Trace, per Section.

Altitudes and Traces

By Martijn Keizer

Overview:

In this tutorial I will try to explain how to make elevation changes in GPL-tracks. Therefore I will start

by writing a little theoretic part about the structuring of altitudes in GPL. This is followed by a close inspection of the formulae that are used in altitude definitions. After this, we should be able to explain the working of the XLS sheets, and how to import and export stuff in there. By then you will have learned enough to start messing around with it in a track, so you might want to have some tips about the use of altitudes on camber, kerbs, roadsurface, hills and so on. This is concluded with a list of DO's and DON'T's, and a list of frequently asked questions on this subject.

Introduction:

Welcome to the Altitude Tutorial. I hope to guide you to being able to modify GPL tracks so that your tracks have nice elevations, banking, kerbs and hills and so on. For this it is necessary to have a little background in the fileformat, for which you will need some basic maths. With a few tips and an explanation of the tools used, I think you should be well on your way to make a track with nice elevations.

Credits:

The whole business of editing tracks the way we do it, would never be possible without the dedication and labour of Phil "Guru" Flack, who started researching the GPL track-file format a long time ago, persisted when others gave up, and now can sit back and enjoy all tracks that are created using his tools, by people who may never know what a hexeditor is for. Furthermore I'd like to thank Paul Hoad for his part in making editors for GP2 and GPL. And last but not least Randy Cassidy, Dave Kaemmer and all the guys at Papyrus that gave us what I consider the best piece of software I've ever used. I could go on and on about how much I love it, but I think the time dedicated to so many people to learn GPL, and make additions to it, gives a much clearer signal.

1. Theory:

In GPL the altitudes of the track are defined by traces. You should see a trace as a line that runs parallel to the centreline of the track. The trace follows its own up-and-down path, independent of other traces. The drivable area is a sort of "filler" between the defined traces. The shape and behaviour of this drivable area is defined by the surfaces and walls, of which there is another tutorial.

The maximum number of traces is 16, while you need at least two. To be able to define every surface and wall, it has to be entirely within the outmost traces. So make sure the maximum lateral distance of a wall is smaller than the distance from centreline of the trace. Trk23dow will crash or fail when you screw this up.

All traces are separately defined for every tracksection. This definition consists of a description of the height*function* of the trace, in the form of a 3rd degree mathematical function, this is a function of the type $\text{altitude} = at^3 + bt^2 + ct + d$ Where T is a counter that goes from zero to 1, depending on the fraction of the section you have already driven.

So to describe 8 traces on a track of 45 sections, you will have to define $45 * 8 = 360$ separate functions. A sequence of a few sections and one trace could look like this, theoretically:

As you can imagine, it's very hard to drive on something like this, as the start and end don't meet up in the first 2 section transitions. The second transition causes some concern as well, as there is a sharp bending point in the track. The car will do a wild bounce probably. Not desirable. The third joint is no problem however. So we can formulate the rule:

In order to have a smooth track with no sudden altitude transitions, the end altitude of one sector needs to be equal to the start altitude of the next sector, AND the end steepness of a sector needs to be equal to the start steepness of the next sector.

Now we go back to dissect the function. With some elementary maths, you can see that;

For $t=0$ (start altitude), altitude = d

For $t=1$ (end altitude), altitude = $a + b + c + d$

The gradient is the first derivative of this height function, so we get the function $\text{Gradient} = 3at^2 + 2bt + c$ where t goes from 0 to 1 This gives:

For $t/dt=0$ (startgradient), gradient = c
For $t/dt=1$ (end gradient), gradient = $3a+2b+c$

Now, remember that this function is relative (in X) to the counter T from 0 to 1. In reality, some sections are longer than others. That means that a mathematical function may be the same on a long or short section, but that the actual sideview of the shorter section, reveals a sort of "compression" in it. This compression causes a higher end-steepness value on the shorter section, relative to the longer section. So we need to bring the section lengths in the calculations to know how steep a section really is (measured in degrees or percentages).

By multiplying the sector length by T, we know the real amount of X-units that the car is from the start of the sector. This way we can calculate the real steepness of the section, by dividing the gradient by the length of the sector. The formulae become like this:

For $t/dt=0$ (startsteepness) steepness (degrees)= $\arctan (c / SL(n))$
For $t/dt=1$ (end steepness), steepness(degrees) = $\arctan (3a+2b+c / SL(n))$
In which SL(n) is the section length of this sector (n)

To be sure two adjoining sectors have a smooth transition, the rules described above can be mathematically formulated like this:

Altitude rule:
 $D(n+1) == A(n)+B(n) + C(n) + D(n)$
In which (n) is the first sector, and (n+1) is the second sector

Steepness rule:
 $\arctan (c(n+1) /SL(n+1)) == \arctan (3a(n)+2b(n)+c(n) / SL(n))$
this can be recalculated to
 $c(n+1) == 3a(n)+2b(n)+c(n) * (SL(n+1) / SL(n))$

Note that now on both equations, the unknowns are on the left, and can be calculated from all the known variables on the right side of the equation.

From this we can conclude that, when you're entering the altitude values of a section, you're not entirely free to do so. The rules of the smooth track surface prove that basically, only 2 out of the 4 variables are user-customizable.

A third degree cubic function is completely defined by a start altitude, end altitude, start steepness and end steepness. As the start altitude and start gradient are defined by the previous section, you would only need to define the end altitude and end steepness in order to give enough data to calculate the parameters. Alternatively, you can define the function completely by using start-altitude and start-steepness, and manually enter the values for A and B, after which a new end-altitude and end-steepness can be calculated.

Now might be a good time to explain parameters A and B to you. Then again, these values are very hard to visualize in a function. If you're familiar with the cubic functions, you'll know that B is the "rate of change of steepness". A is the derivate of this value. Cool huh?

So we have explained the A, B, C and D of the cubic functions. In GPL they are labeled d1 ,d2, d3 and "altitude", respectively. In some witing or sheets you might also encounter d4 and d5. These last two are simply multiplications of d1 and d2, to avoid having to recalculate this number every time (saves frames!).

Now this is a whole lot of calculation going on. Fortunately, we don't have to do that all by hand. There are ways. The method that will be described in the next chapter, uses an Excel sheet to do the calculations.

2. Using the XLS sheets:

2.1 Introduction:

As we have seen in the last chapter, there is an awful lot of parameters that make up the trace data. It's the number of traces * the number of sections * 4 parameters per trace. For a track like Solitude, that totalled to over 7000 parameters. In order to handle these, I made an excel sheet on which I designed all the altitudes. Because its such an extensive excel sheet (of 125 columns and over 1500 rows), this chapter is here to explain to you what the sheet does, how it is set up, and how you can change things on it. This all assumes that you have excel and know how to work with it on a basic level.

2.2 The sheet:

Or rather.. sheets, plural. There are two basic versions of the XLS altitude sheet. One of them edits height using the d1-d2 parameters, and one uses the start-alt / start-grad parameters. Which one you use is a matter of preference. Because you cant have a workable sheet with multiple edited interrelated equation thingies, these two methods are written as two separate sheets.

Load up one of the sheets in excel, and look around a bit. Notice you have multiple worksheets. One is called "*import*", one is called "*Modf*", and one is called "*export*". The purpose of these worksheet is as follows:

Import_data – this sheet is used to copy the output of GTK2xls in. This data can then be read by the other sheets, so you can load a current-state into the sheet and work from there.

Import_result – this sheet displays the resulted calculated d1,d2, start gradient and start-altitude of the centreline of the imported data. It also produces the relative differences in altitude and gradient from this centreline for all the traces. More on that in paragraph 2.3.

A) **Modf_d1d2** – this sheet is used to modify the altitudes. The centreline is defined by d1 and d2, while the start gradient and start altitude are the result of the previous sections. With this method you can quickly update many tracksections if for instance you want the entire track to be 2 metres higher, or if you want to make the altitudes a little more extreme.

B) **Modf_eheg** – this sheet is used to modify the altitude traces, based on the method of defining the end height and end gradient of every section. The start height and start gradient are the result of the previous sections. This method may be a little more comprehensible and intuitive then the previous one, to some. It has the drawback that all changes are done only locally for one section.

A) **Export_d1d2** – this sheet is used to display the results of the settings that you made in the Modf XLS sheet. These results are formatted so that you can easily "copy" the sheet, paste it into a new sheet, save it as CSV and import it into the GTK with Xls2Gtk. Easy.

B) **Export_eheg** – this sheet is used to display the results of the settings that you made in the Modf XLS sheet. These results are formatted so that you can easily "copy" the sheet, paste it into a new sheet, save it as CSV and import it into the GTK with Xls2Gtk.

2.3 Importing, section lengths:

You start out with an empty sheet, where all heigh settings are zero. This structure is flat for the entire track over all the traces. To set up the sheet correctly for inserting altitude data, you have to tell it exactly how long the sections of the track are. As explained in the previous paragraphe, this data is used to calculate what the end gradients are. If you get this data wrong, weird things happen.

Anyway, because we're very lazy, we're not gonne type in 200 9-digit values so we use the export tool GTK2Xls to export a list of parameters out of the GTK. The output of GTK2Xls looks like this:

[GTK2xls output]

In the first column is the sector number. The second column is the length of this sector, in GPL-units (one unit is 1/19685th of a metre) The next columns are d1, d2, d3 and start-altitude for trace 0 of this section, and the next columns are d1, d2, d3 and start-altitude for trace 1, etc. All you need for now, is to copy the sector length column out of the exported data and into the main XLS file (pick one). But

because its just as easy, we rather copy all data from the export to the XLS, the Import worksheet ofcourse..

2.4 Centreline:

In order to limit the number of free variables and make the definitions of the traces more comprehensible, the height of the traces are defined relative to the centreline of the track. That means that you first have to define the up-and-downs of the centreline, and then you can adjust the single traces. Note that this centreline acts as a "virtual trace" now. There doesn't have to be a trace at lateral value zero, but this centreline has the properties of one.

As you define all traces relative to this centreline, its very easy to make ie. a large part of the track a little steeper. Instead of having to do that trace by trace, you can simply make the centreline a little steeper there. Even when all traces get updated at that point, the stucture of that section (banking, kerbs, hills) is retained. So you can see the value of doing it this way, I hope.

2.5 Editing by adjusting d1-d2:

So, now you understand how the centreline concept works, but we're still stuck with an empty spreadsheet. Best way (imo) is to first make the definitions of the centreline, and after that (or a little parallell, most likely) the relative heights of the individual traces.

Personally I like to edit the d1 and d2 values of the centreline, instead of defining the track using gradient and altitude. This is for a number of reasons;

1) I was used to this from GP2 *.

2) Editing d1 and d2 makes it very easy to make changes that affect many sectors, as its possible to change a parameter in the first corner, and that change is calculated through to all the next sections. Ie if you like the altitude sections you've desgined, but have a certain altitude mismatch at the end of the track, you can simply change the first corner to make the entire track 0.1% steeper that will result at the end of the track in perhaps 10m difference, but is unnoticable when driving.

3) D1 and d2 are directly influential on the behaviour of the car. If for instance you make too large a height difference or steepness difference within a (short) sector, the d1 and d2 data will show you relatevely large numbers, so you should be aware then that the car behaviour will be affected more.

So how to edit d1 and d2? Well if you understand what the numbers mean (see chapter 1), it should be possible for you to make some hills and so on. Play around with it a bit. You can always see what the resulting gradient and end-altitude are, in the column in the sheet. You can match this with reallife data if youre fortunate enough to find any. Personally I try to leave d1 to zero, and work only with d2 to make the track go steeper and flatter. The D1 value can be used tho to create altitude effects that you'd otherwise would need to define another section for (like a "step", where the track goes steeper then evens out again, within 1 sector).

Make sure tho that at the end of the track, the end altitude and end gradient have to correspond with the initial altitude and gradient of the track.

2.6 Editing by adjusting gradient and altitude:

Alternative to editing d1 and d2 I made another sheet that allows you to define the traces by giving for every section an end height and end gradient (steepness). The other parameters are then calculated out of the data you enter. The main advantage of this is that it's a little more obvious what youre working with, less abstract. The disadvantages are obviously that one sector is defined independent from the rest and you cant change anything globally. Also it can be hard to spot what exactly causes a certain steep bump that you experience while driving.

To edit the centreline definition using this method, you simply fill in the gradient (in degrees (?)) and the altitude of the track (in metres above "sea level"). Again, make sure that at the end of the track, the end altitude and end gradient have to correspond with the initial altitude and gradient in the first section of the track.

2.7 Editing relative trace data:

So now that we have defined the altitude function of the centreline, we can do the individual traces. This is initially done by entering for every trace, the height of this trace relative to the centreline, at the end of a certain section. The sheet then looks at the centreline definition, grabs the start- and end gradient of the centreline, and uses this for the trace. It also grabs the start- and end altitude by adding the centreline-altitude and relative height together. With these 4 values, the parameters for the height function of this particular trace are calculated.

For example, imagine a banked right turn. All you need to do to make the turn banked, is to make the trace at the left (out)side of the track higher than the trace at the right side. Simply add 20cm of banking to one trace, and -20cm to the other. Note that now, when you change the general altitudes of the track so that this corner becomes uphill or downhill, the banking is unaffected. Take note however that traces can be very widely spaced apart or very close together. Where traces are very close together, the drivable area starts acting like a wall when you put in a lot of height difference between them. You can imagine the situation by drawing a cut-through of the track, with traces at specific points. See picture 2.3

For most cases it is sufficient to define the traces as said. However in some cases you may want to finetune the relative gradient also. For instance on a short banking transition from left to right. For this purpose a section is made with which you can adjust the gradient of a trace relative to the gradient of the centreline. Be careful with this however, for when you enter a large value here, the trace sometimes has to become very steep or curvey to fit the definitions you gave it. See examples in 2.4

<2.4, examples of relative trace gradients>

2.8 Exporting:

Well all this intelligent trace-data generation stuff isn't much use when there is no way to put it all back into a GPL track. That's why we also thought of a "export" function. But because of the use of excel for handling the traces, the exporting function isn't a matter of 2 keypresses, but a few more. In order to get trace data into a GPL track, we use the program Xls2GTK. This program merges a table of data into a GTK file, so you can edit it further. Of course the table of data has to be in a file format that Xls2GTK understands, and that is a CSV (comma separated value) file format, the same as we found when importing in paragraph 2.3. To make this CSV sheet, you simply "copy" the contents of the "export" worksheet from the main Excel sheet, into a new empty sheet. Save this file in CSV format. Remember the name. Now merge this CSV file into the GTK using Xls2GTK. And voila, all of the data you defined in the XLS is now in the GTK. Run Trk23dow to see where you messed up this time around.

2.9 More importing:

This section will be added later. It will describe how to import all altitude data from a GTK into a XLS sheet for further processing.

2.10 Overview:

In pic 2.5 is drawn which functions you have to go through and which way the sheets and programs interact. It's OK if you don't understand this, it's just my weird way of trying to explain simple things with complex unintelligible graphs and words.

2.11 FAQ (F***ing Annoying Questions):

What is Excel?

- It's a Microsoft spreadsheet product that it's pretty useful for, say, calculating altitude functions.

I don't have excel, can you send it to me/ rework the sheets for [insert program here]?

- No. Get excel. Or go away. Or both.

I don't understand any of this as I don't understand excel, and I never used it before!

- Err.. that can be a problem. Ask your friends around for help on this. Don't bother me with simple questions that have anything to do with operating Excel. Making a simple if/then/else or formula is considered required pre-knowledge.

CheckList

Tick everything here and you'll have a complete track !

- <trackname>.TRK - Physical Layout
- <trackname>.3DO - Track Graphics
- <trackname>.CAM - Cameras
- track.ini - Race Control
- *.LP - AI
- *.PBF - Program Covers
- *.3DO - Trackside Objects
- *.MIP, *.SRB - Textures
- horiz.3do+mips - Horizon

- FPS Optimized

trk23dow - Introduction

Introduction -|- .GTK -|- .TSO -|- .TEX -|- .SET -|- .GRV -|- .FB -|- .INI -|- trk23dow.cfg

trk23dow will take a GTK, a bunch of text files giving information on the textures, trackside objects, racing groove, viewing distances etc., and will produce a 3do file (named .n3d to prevent overwriting).

Directories

When you install trk23dow, put it in a directory of its own (e.g. c:\trk23dow), where you will also put all 'compilation' files. These files are .trk files from GPLTrk, all the <trackname>.* files required for trk23dow to work, and any .3do files that you have referenced in your track. This is A Good Idea, because you will then be compiling in one place, not destroying anything you have already made !

In brief, you will go to the dos prompt (start->run... command; change directory to where your track compilation files are), and type:

```
trk23dow -t track > output.txt
```

track is your track name, e.g. monza

the > output.txt part is optional - if you do it, you will get output.txt containing any error messages etc.

Required Files

For trk23dow to work, you must have these files in the same directory:

- <trackname>.gtk- from GPLTrk
- <trackname>.tso- listing the trackside objects
- <trackname>.tex- listing the textures
- <trackname>.set- controls the level-of-detail of the track - number of polygons
- <trackname>.grv- data controlling the racing groove
- <trackname>.fb- forward/backward viewing distances

<trackname>.ini- the same as track.ini

*.3do - any 3do's referenced in <trackname>.tso - these can be in <trackname>.dat or loose in the directory.

trk23dow.cfg - this file tells trk23dow what to output. You can have either a lot (many mb), or a little (a few kb).

Automatic Compilation of Your Track

Nobody wants to go to Dos every time to use trk23dow, so create a .bat file that will do everything at once:

Create a new text document, name it compile.bat (tell Windows where to shove its 'error'). Edit the file so it contains:

```
d:
cd\trk23dow
trk23dow -t mytrack > output.txt
copy mytrack.n3d c:\sierra\gpl\tracks\mytrack\mytrack.3do
```

The first line changes to the drive that you have your copy of trk23dow.

The second changes to the directory.

The third line runs trk23dow on mytrack and outputs to output.txt.

The fourth copies the output .n3d file to your sierra tracks directory and renames it mytrack.3do.

Add any other commands you like, such as copying you TRK file to the GPL directory.

You will have to change the directory names etc.!

Other Functions

trk23dow can do more than just compile your track. By changing the -t on the command line, you can do:

Command	Does
-t	Compiles your track, as explained above
-r	Reverses your track, outputting <trackname>.GK2, with all sections, traces, altitudes and walls reversed.
-f	Fixes altitude angle discontinuities. This smooths some bumps, as well as fixing altitude gaps. Outputs <trackname>.GK2
-z	Zeroes the appropriate values so your track can be used with RPY2LP. Outputs to <trackname>.GK2
-s	Smooooths your track by recalculating all altitudes based on their start/end heights. Outputs to <trackname>.GK2
-x	Flattens your track. Useful if you just want to start the altitudes from scratch. Outputs to <trackname>.GK2

trk23dow - GTK

[Introduction](#) -|- [.GTK](#) -|- [.TSO](#) -|- [.TEX](#) -|- [.SET](#) -|- [.GRV](#) -|- [.FB](#) -|- [.INI](#) -|- [trk23dow.cfg](#)

<trackname>.GTK

trk23dow does not like all GTKs. This is because not all GTKs you can create with GPLTrk contain

sufficient detail for trk23dow to process them correctly. Here you will find detail on the specific requirements, and the options available to control the way your GTK is processed.

Traces

trk23dow will not create any polygons before the first trace, or after the last trace. In other words, the first and last traces are clipping planes. The reason for no polygons is that trk23dow does not have any altitude data to work with.

Sections

Sections cannot be ≥ 180 degrees in curvature. I strongly suggest making 90 degrees your limit; splitting sections of greater curvature into 2.

You must make sure that your sections do not have gaps between them. Although this won't stop trk23dow working, it will cause GPL to lock up when you drive *on* the 'gap' (doesn't happen every time you drive over the gap if the gap is small). The [Creating a TRK](#) page has more detail on gaps.

Altitudes

Your altitudes can be anything - trk23dow will process them. If you want your track to look good, however, you should try to keep your altitudes the same across all traces, per section. Exceptions such as hills, banking, etc. should be accompanied by additional polygons for the walls that cover those altitudes. Use unk4 flags to increase the number of polygons (or even decrease them so that one polygon covers all - it often works !)

Walls

Walls cover the entire track surface. trk23dow creates polygons for every wall in your track. The number/texture/style of polygons is controlled using unk4 flags, listed below. Note that any combination of flags can be added together, although this may cause weirdness, especially if you use flags together that do the same/opposite function.

[FWO] means First Wall Only - you should only use this on Wall #0 of a section - it will have no effect when applied to any other walls.

For detail on the TEX Settings, see the [TEX page](#).

Flag number	Explanation
1	Inverts texture coordinates left-right. Untested - a relic of older versions
2	Where a wall covers/crosses more than one trace, this will stop trk23dow from splitting the wall on those crossings. This also forces the texture to be mapped once left-right. Do not use when... the altitudes corresponding to the covered traces are non-planar - e.g. a cambered piece of track. Do use when... you have a flat bit of track and you are saving polygons. See also - flag 512
4	Creates this wall from triangles instead of quads. However, if a polygon is found to be non-co-planar, it will automatically be created from triangles.
8	The texture for this wall will be mapped once, stretching to fit. This is useful for things like track markings (e.g. the markings at the beginning of a runway).
16	When vertical walls are created, by default they do not have polygons created for the inside-base - where the ground level would be if the wall was not vertical. This flag will create those polygons for this wall.
32	For walls with height, this flag adds polygons to the beginning (front). TEX Setting: 4. See also - flag 64.
64	For walls with height, this flag adds polygons to the end (back).

	<p>TEX Setting: 5. See also - flag 32.</p>
128	<p>Creates upside-down polygons, just under the surface of the track. This is used when you need to see the underside of the track - e.g. a bridge/crossover. TEX Setting: 3.</p>
256	<p>[FWO] This setting stops trk23dow from splitting any of the walls in this section based on the values in <trackname>.set Do not use when... you have a hilly piece of track, or you have varied altitudes left-right, or you have walls that cross several traces. Do use when... you have a flat piece of track, or a constant gradient piece of track. This will save FPS. See also - flags 2048, 4096.</p>
512	<p>Exactly the same as flag 2, except this does not stop the wall from being split where it crosses traces. Do not use when... it makes things go weird. Do use when... flag 2 messes up, leaving gaps, but you still require the texture to be mapped once. Often used for the track wall. See also - flag 2</p>
1024	<p>The entire track is covered with visibility boxes. In normal circumstances they are created between walls with height. E.g. one from the right extremity to the pit wall, one for the pit wall, one from the pit wall to the armco on the left, one for the armco, one from the armco to the left extremity. However, when your track has elevations such as banking, you will likely see clipping effects due to the polygons being drawn in the wrong order. Add 1024 to the flags of the wall immediately to the right of where the problem is - e.g. add it to the wall that is the banking. The problem should go away. It will not be solved on all corners though, due to evil stuff :- </p>
2048	<p>[FWO] Increases the number of polygons for every wall in this section by 5, longitudinally. Do not use when... you have bad FPS, or the section is a corner (5*4 = 20 times as many polygons = bad!) Do use when... the cars appear to either drive under, or float above, the track surface. Also, you can use on corner sections when they appear too blocky, although 4096 is better for corners. See also - flags 4096, 256</p>
4096	<p>[FWO] Increases the number of polygons for every wall in this section by 2, longitudinally. Do not use when... you have bad FPS, or the section is a corner (5*2 = 10 times as many polygons = bad!) Do use when... the cars appear to either drive under, or float above, the track surface, or a corner is too blocky. See also - flags 2048, 256</p>
8192	<p>[FWO] This will increase the number of groove polygons in this section by performing a longer calculation on the groove data for this section. Do not use when... you have bad FPS, or the groove is already ok in appearance. Do use when... the racing groove is too blocky, goes all over the place. See also - flag 16384.</p>
16384	<p>[FWO] Creates one groove polygon for the entire track section - the complete opposite of flag 8192. Do not use when... it makes the groove too blocky/weird. Do use when... the racing groove is too smooth and you get bad FPS. See also - flag 8192.</p>
32768	<p>This stops trk23dow from creating a polygon for the back-facing side of walls with height. This is useful for when the back-facing sides of walls will never be seen because of the track layout. Back-facing is defined as: left-facing if the wall is to the left of the centre-line; right-facing if the wall is to the right of the centre-line; undefined if the wall is on the centre-line.</p>

65536	[FWO] By default, corner sections are created in three resolutions - high (4 pieces), medium (2 pieces) and low (1 piece). Setting this flag will create the section from high (2 pieces), medium (1 piece) and low (1 piece). This has two main effects: 1) Decreases the number of polygons, thus improving FPS. 2) Decreases the number of boxes covering the track (see flag 1024), thus improving FPS, but decreasing the graphical quality. See also - flag 1024.
131072	This stops trk23dow from creating top polygons for walls with height - e.g. the top of trees that you can't see. This is a FPS saving measure.
262144	[FWO] This will add ceiling polygons for all walls in this section. The height above the ground-level polygons is taken from the height of the first wall, which will normally be the side of a tunnel or something similar. TEX Setting: 6 See also - flag 524288
524288	This turns off ceiling polygons for this section - e.g. the walls that are the side of the tunnel. See also - flag 262144.

trk23dow - TSO

[Introduction](#) -|- [.GTK](#) -|- [.TSO](#) -|- [.TEX](#) -|- [.SET](#) -|- [.GRV](#) -|- [.FB](#) -|- [.INI](#) -|- [trk23dow.cfg](#)

<trackname>.TSO - Track Side Objects

The .TSO file is used to position Track Side Objects. These are all the buildings, flaggers, trees, etc. that you see around the tracks.

Here is an example TSO file:

```
# Trackside 'Sets': 51
Set List
[ 0]: 3, 170.0, -7.0, 0.0, 0.0, 0.0, 0.0, 1.0, sign100 braking board for
corner1
[ 1]: 3, 120.0, -7.0, 0.0, 0.0, 0.0, 0.0, 1.0, sign050 braking board for
corner1
[ 2]: 3, 90.0, -10.0, 1.0, 0.0, 0.0, 0.0, 1.0, signshe shell
advertisement
[ 3]: 3, 383.0, -7.0, 0.0, 0.0, 0.0, 0.0, 1.0, sign050
[ 4]: 3, 343.0, -8.0, 0.0, 0.0, 0.0, 0.0, 1.0, signshe
[ 5]: 3, 243.0, -7.0, 0.0, 0.0, 0.0, 0.0, 1.0, sign200
[ 6]: 3, 1420.0, -6.0, 0.0, 0.0, 0.0, 0.0, 1.0, sign100
[ 7]: 3, 1320.0, -6.0, 0.0, 0.0, 0.0, 0.0, 1.0, sign200
[ 8]: 3, 1878.0, -6.0, 0.0, 0.0, 0.0, 0.0, 1.0, sign100
[ 9]: 3, 1928.0, -6.0, 0.0, 0.0, 0.0, 0.0, 1.0, sign050
[ 10]: 3, 10.0, -5.0, 0.0, 0.0, 0.0, 0.0, 1.0, starter green flag man
[ 11]: 3, 1839.4089, 0.0, -0.1016, 180.0, 0.0, 0.0, 1.0, mbridge maxim
bridge
[ 12]: 3, 2235.0, -14.5, 0.0, 270.0, 0.0, 0.0, 1.0, garages
..
..
..
2 InfieldObject
[ 48]: 3, 72.0, -10.0, 0.0, 285.0, 0.0, 0.0, 1.0, pit
5 Reference
{ 0}: 0
{ 1}: 1
{ 2}: 2
```

```

{    3}: 220
{    4}: 221
[   49]: 3, 7260.0, -18.0, 0.0, 0.0, 0.0, 0.0, 1.0, tent
1 Reference
{    0}: 219
1 OutfieldObject
[   50]: 1, -14.382953, -1397.6690, -406.908, 0.0001, 0.0001, 0.0001, 1.0,
tower1
3 Reference
{    0}: 98
{    1}: 99
{    2}: 100

```

The first two lines tell trk23dow how many objects are in the first section (green). These objects are those that are within the driving limits of the track. Each line in the first section consists of a number of values. Here is the entry number 10:

```
[   10]: 3, 10.0, -5.0, 0.0, 0.0, 0.0, 0.0, 1.0, starter green flag man
```

[10]: - this is the object number, starting from 0.
3, - this is the method of object placement (see Trackside Objects for more detail)
10.0, -5.0, 0.0, 0.0, 0.0, 0.0, - this is the position and rotation of the object (see Trackside Objects for more detail)
1.0, - this is a scale value. It doesn't seem to work as expected
starter - this is the name of the object. In this case it is the bloke who waves the flag at the start of a race.
green flag man - this is a comment. Everything from the end of the object name (end is defined by a space), to the end of the line is deemed a comment.

The second section (blue) is the objects that are to the right of the track, outside the driving limits. (Infield assumes a clockwise track. If you have an anti-clockwise track, this is actually the outfield, but is still called the infield).

The lines are the same as before, but after each line you must have a number of TRK Section ID References:

```

5 Reference
{    0}: 0
{    1}: 1
{    2}: 2
{    3}: 220
{    4}: 221

```

This example means that this object (the one named 'pit'), is situated next to the TRK sections 0, 1, 2, 220 and 221 (it crosses the s/f line).

'next to' means that as you look at the TRK, this object is to the right of the centreline, outside the driving limits.

The third and final section is the objects that are to the left of the track, outside the driving limits. (Outfield- clockwise).

The system of references is the same as for the Infield objects.

See TrackSide Objects for further information.

Do Not !

Use object names or comments that begin with a number or symbol - always start with a letter.

Don't use long file names - GPL does not like them, neither does trk23dow.
Deviate from the layout of the TSO file. Blank lines will cause problems as well.

trk23dow - TEX

[Introduction](#) -|- [.GTK](#) -|- [.TSO](#) -|- [.TEX](#) -|- [.SET](#) -|- [.GRV](#) -|- [.FB](#) -|- [.INI](#) -|- [trk23dow.cfg](#)

<trackname>.TEX - Texture Mapping

In order for trk23dow to map the right textures to the right polygons, you must create a <trackname>.TEX file that relates to the values specified in your TRK.

Surface Subtypes

By now you should understand the different TRK surface types, but this will only let you define 1 texture per surface type - hardly a graphically pleasing track. To overcome this, the Unknown3 value in the TRK Walls is used as a surface SubType value, letting you have e.g. Asphalt 0, Asphalt 1, Grass 42, Armco 4948583, or whatever (maximum value of $2^{32}-1$).

That .TEX File Then...

Here are the first few lines from a .TEX file:

```
0 0 0 10.0 10.0 error
1 0 0 10.0 10.0 asphalt
1 1 0 10.0 10.0 road
1 2 0 10.0 10.0 run
1 3 0 10.0 10.0 asf_shw
1 4 0 10.0 10.0 asp_sl
1 5 0 10.0 10.0 asp_sr
2 0 0 15.0 15.0 dirt
2 0 1 20.0 20.0 dirt
2 0 2 20.0 20.0 dirt
2 0 4 20.0 20.0 dirt
2 0 5 20.0 20.0 dirt
```

The first number in each line is the surface type - 1=asphalt, etc.

The second number in each line is the surface subtype - the value corresponding to the Unknown3 value in the TRK.

The third number is the 'side':

- 0 = normal texture, mapped on to the top of the wall
- 1 = vertical surface - e.g. side of armco
- 2 = upper ground-level polygon - e.g. top of armco
- 3 = ground-level, reflected - for when the track is visible from underneath.
- 4 = front of the wall (longitudinally)
- 5 = back of the wall (longitudinally)
- 6 = ceiling - e.g. the inner-top of a tunnel

The fourth number is the longitudinal scaling value - e.g. the asphalt texture will be mapped every 10.0 metres.

The fifth number is the lateral scaling value - (left-right).

The last part is the MIP name to use.

Note the first line - this is 0 0 0 10.0 10.0 error. Here you should define a texture to be used in case you forget to define a texture for any surface.subtype combinations. Make it a texture that you can easily spot in GPL, so you can work out where the problem lies.

See how surface type 2, subtype 0 has five textures mapped to it. This is because somewhere in the TRK, surface 2, subtype 0 has been given height, to make it into a vertical wall.

If you use any of the Unknown4 flags to change the texture mapping, the scale values defined in the .TEX file are overridden.

trk23dow - SET

[Introduction](#) -|- [.GTK](#) -|- [.TSO](#) -|- [.TEX](#) -|- [.SET](#) -|- [.GRV](#) -|- [.FB](#) -|- [.INI](#) -|- [trk23dow.cfg](#)

<trackname>.SET - Detail Settings

This little text file gives you some control over the number of polygons generated for your track. It is now mostly replaced by [TRK settings](#), but you should still know what this file does.

100.0 - value A (metres)
300.0 - value B (metres)
200.0 - scale A (metres)
200.0 - scale B (metres)
200.0 - scale C (metres)
20.0 - no longer used
0.10 - corner error (metres)
1000 - initial value, leave alone!
1000 - initial value, leave alone!
100 - initial value, leave alone!

What does it all mean ?

For TRK sections of length 0.0metres to value_A metres, scaleA determines what the maximum length of each polygon is.

Examples:

valueA=100.0metres, scaleA=20.0metres, you would get a maximum of 5 polygons longitudinally in sections less than 100.0metres in length.

valueA=100, valueB=300, scaleB=50.0metres, you would get a maximum of 4 polygons longitudinally in sections between 100 and 300 metres in length.

valueB=300, scaleC=200, you get one polygon per 200 metres longitudinally in sections longer than 300 metres.

Corner Error is a value that controls how many times each corner is split. It is a value in metres, so 0.10 is 10 centimetres - this is the maximum deviation from the true circle that the polygons will be. This error value is calculated at the centre of the track ! In other words, there will be greater error on the outside of corners to the inside.

Typically, a value of > 0.10 gives a blocky track, a value of 0.05 gives a smooth track, and a value of 0.01 gives a track that is **very** smooth.

trk23dow - GRV

[Introduction](#) -|- [.GTK](#) -|- [.TSO](#) -|- [.TEX](#) -|- [.SET](#) -|- [.GRV](#) -|- [.FB](#) -|- [.INI](#) -|- [trk23dow.cfg](#)

<trackname>.GRV - Racing Groove

This is a file that tells trk23dow where to create the racing groove, and what darkness to make it.

Example file:

```
60sections.122entries
1 2.388616 0.0 130.5 3.0
1 2.382926 0.0 131.0 3.0
1 2.377237 0.1 131.5 3.0
1 2.371547 0.1 132.0 3.0
1 2.365858 0.1 132.5 3.0
1 2.357933 0.1 133.0 3.0
1 2.350008 0.1 133.5 3.0
1 2.342083 0.1 134.0 3.0
1 2.335848 0.1 134.5 3.0
1 2.329612 0.2 135.0 3.0
```

The first line does not matter: it could say "this is my groove file" - it is just a line for the editor-person. The rest of the lines are in this format:

- Number
- Lateral Position in metres
- Darkness setting from 0.0 (invisible groove) to 1.0 (solid groove). Setting sections to 0.0 will stop trk23dow from creating groove polygons for those parts of the track (FPS saver!).
- Longitudinal Position in metres - distance from the start of the track.
- Lateral width of the groove in metres. This lets you create fancy things like a darker section covering the grid boxes, for that added touch of realism. The groove is centred on the lateral position, so half of the lateral width is each side of that.

Creating a GRV from a Replay

<Matt Mini Tutorial Required>

trk23dow - FB

[Introduction](#) -| [.GTK](#) -| [.TSO](#) -| [.TEX](#) -| [.SET](#) -| [.GRV](#) -| [.FB](#) -| [.INI](#) -| [trk23dow.cfg](#)

<trackname>.FB - Forward/Backward Viewing Distances

When you drive around your track, you want to be able to see everything that you would see in real life, but you don't want to see everything that is hidden around the corner, or even everything behind you (to maximise FPS).

The .FB file lets you specify the distances from the current viewpoint, in metres, that you will be able to see. Here is a small extract:

```
400.0 250.0 0.0 0.0
385.0 250.0 0.0 0.0
370.0 250.0 0.0 0.0
355.0 250.0 0.0 0.0
340.0 250.0 0.0 0.0
320.0 250.0 0.0 0.0
280.0 250.0 0.0 0.0
240.0 250.0 0.0 0.0
225.0 250.0 0.0 0.0
210.0 250.0 0.0 0.0
```

There is one line in the .FB file for every 128feet (39.0144metres) of the track length. You should include a few extra values at the end.

- **First Value:** Distance ahead that you can see, in metres. This is the 'normal' viewing distance
- **Second Value:** Distance behind that you can see, when driving forwards - i.e. mirror viewing distance
- **Third Value:** Distance ahead that you can see when you're going the wrong way.
- **Fourth Value:** Distance behind that you can see when you're going the wrong way.

The Third and Fourth values are also used when you can see the other way up the track - e.g. when approaching a hairpin and you can see where the track goes after the turn. There must be some minimum value at work here, because you can use 0.0.

Also note that these values are affected by the detail slider in the GPL options menu.

An Example

When creating your .FB file, you can keep the values small by considering every point on the track individually - e.g.:

On the approach to a blind hairpin (lots of trees on the infield), you cannot see anything beyond the hairpin until you turn round the corner. In this example, you can reduce the Forward viewing distance so that nothing beyond the hairpin will be drawn. Right up near the hairpin, the Forward viewing distance will be 100metres at most (to cater for any objects on the outfield).

trk23dow - INI

[Introduction](#) -|- [.GTK](#) -|- [.TSO](#) -|- [.TEX](#) -|- [.SET](#) -|- [.GRV](#) -|- [.FB](#) -|- [.INI](#) -|- [trk23dow.cfg](#)

track.ini

This file is track.ini renamed to <trackname>.ini, e.g. goodwood.ini. trk23dow requires it to position the grid boxes on the track, nothing more, nothing less. See the [track.ini](#) guide for information about the format.

trk23dow.cfg

[Introduction](#) -|- [.GTK](#) -|- [.TSO](#) -|- [.TEX](#) -|- [.SET](#) -|- [.GRV](#) -|- [.FB](#) -|- [.INI](#) -|- trk23dow.cfg

trk23dow.cfg

This file contains a few 0/1 options that let you change how much output trk23dow produces. **WARNING !** If you set everything to 1, it is unlikely that you will be able to load the output file because of its size ! A default trk23dow.cfg is included with the trk23dow download.

Options

```
1 ;slu output
0 ;trackloading
0 ;tracksaving
0 ;trackside objects
0 ;polygons
0 ;texture stuff
0 ;*serious* amount of output
0 ;groove related
0 ;grid/sf line related
```

```
0 ;extremities listings
0 ;strings
1 ;sound
0 ;lists (vertices, planes, etc.)
0 ;t15 stuff
0 ;node tree listing
```

slu output = section look up; this is how corners are broken into smaller pieces

trackloading = output information when loading a track.trk/.3do

tracksaving = output information when saving a track.trk/.3do

trackside objects = output information about the trackside objects referenced in <trackname>.tso, where they are placed...

polygons = information relating to the polygons generated (lots of polygons get generated!)

texture stuff = exactly as it says- textures, texture coordinates

serious amount of output = everything else, which can be quite a lot

groove related = surprisingly, this is the information relating to the racing groove (<trackname>.grv)

grid/sf line related = the sf line is created right at the end of the track, the grid boxes are created according to <trackname>.ini

extremities listing = an output of the coordinates at the extreme left and right of your track

strings = all strings used, e.g. .mip names, .3do names

sound = sadly no longer with us due to the port to windows :(aaah, the good ol' days when trk23dow could rival a Spectrum loading symphony :-)

lists (vertices, planes, etc.) = big lists, fairly boring

t15 stuff = t15 boxes cover the entire track and are used for the trackside objects. Too many will slow your track down, too few make it look crap in the corners.

node tree listing = every polygon in your track hangs off a big tree structure (that is actually circular in places - go Papyrus! ;)

Getting Track Information

People

Don't live near the track ? Find someone who does ! There are usually people who live near the track you have chosen and who are willing to do a bit of researching for you - just make sure you include them on your beta-team !

Maps

Before you start making a track, it is important that you have an accurate map to work from. 1:10000 scale is good enough, but you really want 1:5000 or better. Most countries have some form of national mapping agency (e.g. the UK has the Ordnance Survey: <http://www.ord-svy.gov.uk/>) so see what you can get from them. You can often get aerial photography or digital mapping, but these are usually very expensive for what they are (and what the project is...)

Centre Line

You want to make sure your centre line is as accurate as possible. There are several ways you can do this:

- 1) Become a draughtsman and use accurate rulers, compasses, protractors etc., to take measurements directly off the map (a photocopy preferably).

- 2) Use Paul Hoad's F1GP2 Track Editor to trace over a scanned map, placing straights and corners as you go. This can work very well, although the maximum image size you can use is a bit limited for larger tracks.

3) Trial-and-error approach. For twisty sections of track, it is very tricky to make accurate measurements. Sometimes it is best to create the track as best you can, then compare it with a known map, and adjust the offending sections.

At the end of the day, you want a track that matches a real map accurately, and that has a tracklength the same as the real track (to the nearest 5 metres is considered good).

Track Features

For the rest of the track, you are unlikely to find a map that will show all the track details - hedges, kerbs, run-off areas, etc. You will get maps showing access roads/runways, and the edges of the track, but for everything else you will need photos/film/local knowledge/imagination.

Altitudes

At 1:10000 scale, you usually get 5metre interval contours and the occasional spot-height. This is enough, so long as you also have knowledge of the 'local' elevations - bumps, camber, etc. Hopefully you have an in-car lap on video and from it you can see where the bumps are. Camber is quite tricky to spot, but a good track will always have some camber in every corner, because it makes it more fun !

The goal for altitudes is to have a 'global' set of altitudes that will give you the general vertical shape of the track, and a 'local' set of altitudes that lets you create the camber, bumps, banking, verges, hills, volcanoes etc.

Photos/Film

If you can, get a video of an in-car lap around the track. Better still, go to the track and video the lap (slow lap !) If you visit the track, make sure you get plenty of photographs of the track, the track surroundings (banking, trees etc.), and the buildings- for textures & 3D models.

Track Camera settings by Ed Solheim [adapted for TrkCAMEdit by Joachim Blum]

After having edited a few of the replay cameraset at some GPL tracks, this is what I know about the camera-formats. I begin with what I know about the format for the TV1- and TV2-cameraset

NOTE: this does NOT apply for the pitlane-camera which uses a different format and will be explained later:

TV1- and TV2-cameraset:

Camera value #1:

Camera position argument #1 - The distance the camera is located from the start-/finish-line of the track in meters.

Camera value #2:

Camera position argument #2 - The distance the camera is located from the defined center-line of the track in meters. A positive value puts the camera on the left side of the track and a negative value puts the camera on the right side.

NB: I say center-line and not center of the track, because at some tracks the defined center-line and the center of the track are not always the same. Edit a camera located at the start-/finish-line at Monza and set value #2 to "0" and you will see what I mean.

Camera value #3:

Camera position argument #3 - The height the camera is located from the ground surface in meters.

Camera value #4:

Camera change-over point - Value (meters) tells the GPL "director" when to switch over from the previous camera to this one. In other words: When the car is within this distance from the camera position, then the camera will be activated. A negative value activates the camera before the car reaches the camera position, a positive value activates the camera after the car has passed the camera position.

Camera value #5:

Camera-type - a "0"-value means that the camera will follow the car and a "1"-value will lock the camera. It seems that setting the value to 1 makes GPL use camera values #9 to #12 for zoom and picture location. More about this later. In TrkCAMEdit, simply check or uncheck the option to choose between these two camera-types.

Camera value #6:

Camera zoom #1 - Initial zoom value. This is the amount of zoom the camera will use when it is activated. This value is not used by fixed cameras. Unit is still unknown.

Camera value #7:

Camera zoom #2 - Zoom value for first part. The value sets the amount of zoom the camera will use from the time it is activated until the car reaches the camera position. This value is not used by fixed cameras. Unit is still unknown.

Camera value #8:

Camera zoom #3 - Zoom value for last part. The value sets the amount of zoom the camera will use from the time the car passes the camera position until the switch-over to the next camera. This value is not used by fixed cameras. Unit is still unknown.

Camera value #9:

Fixed camera focus orientation #1 (Yaw) - This value tells GPL where to point fixed cameras at. A positive value will point the camera to the left in relation to its position and a negative value will point it to the right. A "0"-value will point the camera parallel to the centerline of the tracksection where it is positioned, looking into the normal driving direction. The original unit is radians, however, TrkCAMEdit uses degrees for this value.

Camera value #10:

Fixed camera focus orientation #2 (Pitch) - This value tells GPL where to point fixed cameras at. A positive value will rotate the camera downwards in relation to its position and a negative value will rotate it upwards. The original unit is radians, however, TrkCAMEdit uses degrees for this value.

Camera value #11:

Fixed camera picture orientation (Bank) - Tells GPL what way to rotate the picture (i.e landscape or portrait like). A positive value will rotate it clockwise and a negative value will rotate it counterclockwise. At 180 degrees the picture will be upside-down. The original unit is radians, however, TrkCAMEdit uses degrees for this value.

Camera value #12:

Fixed camera *POV* (Point Of View)- This value sets the opening angle (or otherwise: the zoom) for fixed cameras. The lower the value the more it will zoom-in. The original unit is radians, however, TrkCAMEdit uses degrees for this value.

Camera value #13:

UNKNOWN!

Camera value #14:

UNKNOWN!

As you can see, it seems like the moving cameras only use values #1 to #8 while fixed cameras use values #1 to #5 and #9 to #12. Because of this, TrkCAMEdit will disable the not used values according to the setting of value #5 (checked or unchecked) as kind of a "hint" for the user.

As of now I do not know what values #13 and #14 do, they all seem to be set to "0" for all cameras at all tracks.

Pitlane camera:

The pitlane-camera uses completely different values than the TV1- and TV2-camerasets. So far this is what I know about it:

Camera value #1:

Camera position #1 - longitudinal position in meters from the center of the whole track (that means, the center of the global coordinate system used).

Camera value #2:

Camera position #2 - lateral position in meters from the center of the whole track (that means, the center of the global coordinate system used).

Camera value #3:

Camera position #3 - height above track/surface in meters.

Camera value #4:

Pitlane-camera focus orientation #1 - Yaw. Similar to value #9 for the fixed cameras. A positive value will point the camera to the left in relation to its position and a negative value will point it to the right. A "0"-value will point the camera parallel to the centerline of the tracksection where it is positioned, looking into the normal driving direction. The original unit is radians, however, TrkCAMEdit uses degrees for this value.

Camera value #5:

UNKNOWN! Editing this value does not seem to have any effect at all.

Camera value #6:

Pitlane-camera picture orientation - Bank. Similar to value #11 for the fixed cameras. Tells GPL what way to rotate the picture (i.e landscape or portrait like). A positive value will rotate it clockwise and a negative value will rotate it counterclockwise. At 180 degrees the picture will be upside-down. The original unit is radians, however, TrkCAMEdit uses degrees for this value.

Camera value #7:

Pitlane-camera POV. Similar to value #12 used for the fixed cameras. This value sets the opening angle (or otherwise: the zoom). The lower the value the more it will zoom-in. The original unit is radians, however, TrkCAMEdit uses degrees for this value.

Camera value #8:

For some strange reason... editing this value seems to have the same effect as #4 - i.e. camera focus orientation #1 - yaw. Why these do the same - I have no clue ;)!

Camera value #9:

Pitlane-camera focus orientation #2 - Pitch. Similar to value #10 for the fixed cameras - with one exception - it is reversed! A negative value will rotate the camera downwards in relation to its position and a positive value will rotate it upwards. The original unit is radians, however, TrkCAMEdit uses degrees for this value.

Camera value #10:

UNKNOWN! Editing this value does not seem to have any effect at all.

Camera value #11:

UNKNOWN! Editing this value does not seem to have any effect at all.

Camera value #12:

UNKNOWN! Editing this value does not seem to have any effect at all.

That is about all I know at the moment. If you got any additional information that I have missed or maybe other explanations then please feel free to drop me a line or 3. The address is as always: ed@gplea.org

Hope this helps a few souls out there and that you all have a good time editing new tracks and cameras!

Making it all look good - the business of textures

[section 2]

Overview

You can make a new GPL-track as great as Alexander the Great, but without good looking textures you end up with something looking rather kack. The textures in GPL are small files based on bitmaps called 'mip' (Multiple Image Pictures). These images are 'mapped' or displayed on objects in various ways, governed by lots of rules.

Making the images work in GPL is quite easy due to the excellent tools developed by Phil, Nigel and Klaus, but the hard part is the overall design and layout of things - the true art. The guy responsible for textures needs to work in total cooperation with the track/3do makers, and do some Bossing around. There are lots of things to consider, and plenty of ways to do this, so I'll do this from a rather simple view: my own. Many talented artists are out there, and there will be faster/better ways of doing this, but I enjoy it, it's a real joy to sit back and see it all come together after starting out as a flat piece of grass/asphalt! This is the most rewarding part of track creation.

How is it done?

As I said; making the mip is easy. The process, explained in few words:

- Draw the texture
- Save it as a .bmp in the chosen number of colors.
- Using one of the mip tools, make the mip from the bmp, at the same time deciding repeating value and transparency
- Make a track or a 3do and view it :o)

Defining the MIP

A Mip has certain properties, they are all important, and they must all follow specs. Here are the important ones:

- Size
- Texture
- Transparency
- Repeating value
- Colors

Planning ahead

Always try to plan ahead: can you use this mip in several locations? How does that affect the properties? How does the brightness and contrast fit in with the rest of the graphics? Is the pure texture size optimized?

The best way to start out, is by deciding the seasoning from the start. Think about the

following:

- Which time of year is this? Should the trees have leaves? Yellow grass? Flowers? (oooh!)
 - Is the circuit & surroundings worn, or perhaps everything was brand new in the day you're replicating?
 - What's the weather like? Is it sunny? Overcast? Wet? Blizzard?
 - Are there much 'dark' areas? Shadows?
 - What kind of vegetation is right for the area? No palm trees in Scandinavia!
-and so on. You get the idea.

Size

You've probably seen the excellent dashboards from the GPL Spares Factory. The reason they look so highly detailed, is that huge bmp's (and therefore mips) are used. Well, this gives a new challenge: how does this affect GPL?

Using these mips (their size, in pixels, is 512x256) will not work on 3Dfx graphic cards because they simply don't support textures bigger than 256x256. So keep this in mind. The size always has to be the power of 2 for a mip to work, here are the most common sizes: 256x256, 256x128, 256x64, 256x32, 256x16, 128x256, 128x128, 128x64, 128x32, 128x16, 64x256, 64x128, 64x64...and so on. Any deviation from these sizes, and the mip tools won't work, or if you should manage it anyway, you'll get the dreaded 'Could Not Load Track' error in GPL.

There is a catch - the bigger textures you use, the more energy they suck out of your gfx card, and frame rate drops. For instance, if you're making a wall, try not to use one huge mip, but rather small, repeating ones, using *bump-mapped* textures. More on this later. But using a small texture will look bad when close-up because it gets pixellated, there is too little detail. This is something we just have to live with in these days. Not quite real-life just yet!

Repeating Value

You will want to use repeating mips, this is coded in the mip itself when using one of the tools. A mip can either be told to repeat sideways (nice for a barrier), upwards (telephone pole), or both (brickwalls on a building). This is how you can get away with the grass and asphalt; the same texture repeats itself and fills that wall in your track.

TIP! *The 'stretch' or sizing of the textures is set in the .tex file. If you have a barrier, and set it to stretch 10.0, this will look fine on the straights, but awful in corners. Fix this by having another tex entry using the same mip, but with a smaller repeating value calculated roughly by the radius of the turn, example 7.0.*

The repeating value is also governed by the unk 4 values in GPLtrk, you can tell trk23dow to repeat it once from left to right, once the entire section, etc etc. read the section on walls to learn.

If you have set a repeating value for a mip, and it still looks odd, try making it repeat the other way, and then check the GTK for the unk 4 value if it is a texture on the track itself.

Colors

The original GPL mips are 99% 4-bit mips, that means a palette of 16 colors or less for each texture. These are good for FPS, but often can look like gnorphywooba. You can make great looking 16-bit mips (woohoo, millions of colors) but these will take up some resources on the gfx bord, and on certain cards may not look to good, become blurry and so on. So do some testing. For some textures, 16-bit is the only way out.

D3D cards, like the GEForce, converts all textures to 16-bit anyway, and with 3Dfx bought out (sob!), the number of colors may not matter that much in the future. But try and make them in 4-bit, you will be surprised how good they look in GPL, and performance is great. Later I will explain how you can make a 4-bit texture look snazzy!

Transparency

There is a lot of possibilities working with variable transparencies in 16-bit mips in conjunction with polygon colors and so on, but I have yet to experiment with that, save for the groove mip. Tip: get a nice groove from somebody else!

You are allowed one transparent color in a mip, this is set in the tools. For the 16-bit mips, you'll also need to make a texture in 2 -colors which works as a transparency map. More on all this when I try to explain the tools.

Let's make a mip, and go through the process of using the tools.

Making a mip from scratch

[section 1]

As an example, I'll make a 16-bit mip (this is with loads of colors), and it is intended for a wall with height, and will have transparency. The texture is the ruin wall to the left of the pits at Crystal Palace.



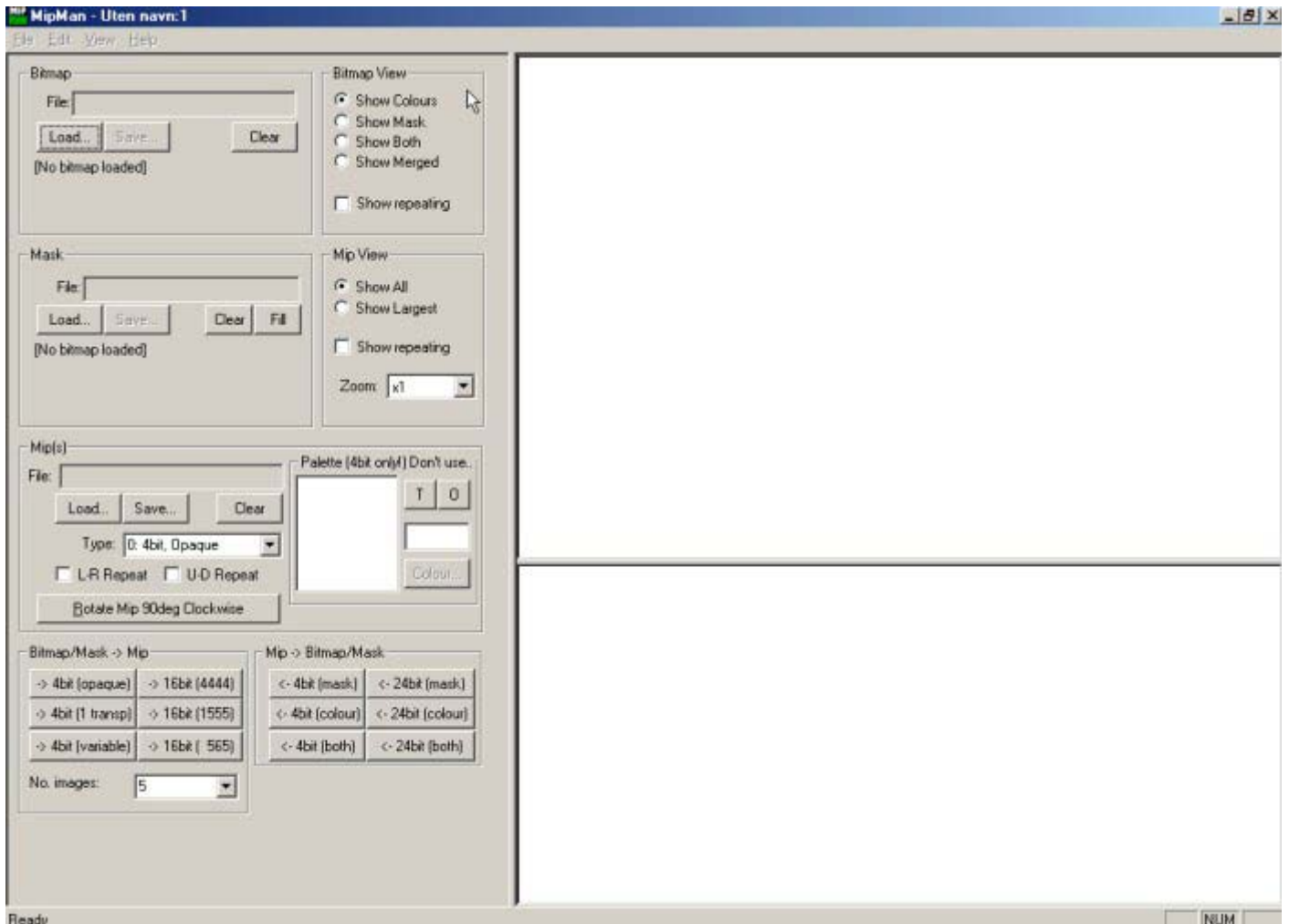
Here is the bitmap. The size I've chosen is 256x128 pixels, designed to repeat sideways. Everything except the people are hand drawn in Paint Shop Pro 6. (Head over to www.jasc.com to get a free demo). You can cheat a bit with having a nice stone texture to start with.

So I had to make this mask for the mip, the black being the transparent area. To make the mask, I usually mess around with brightness/contrast, reducing colors and so on, until you end up with the right look, then fill with white and black respectively.

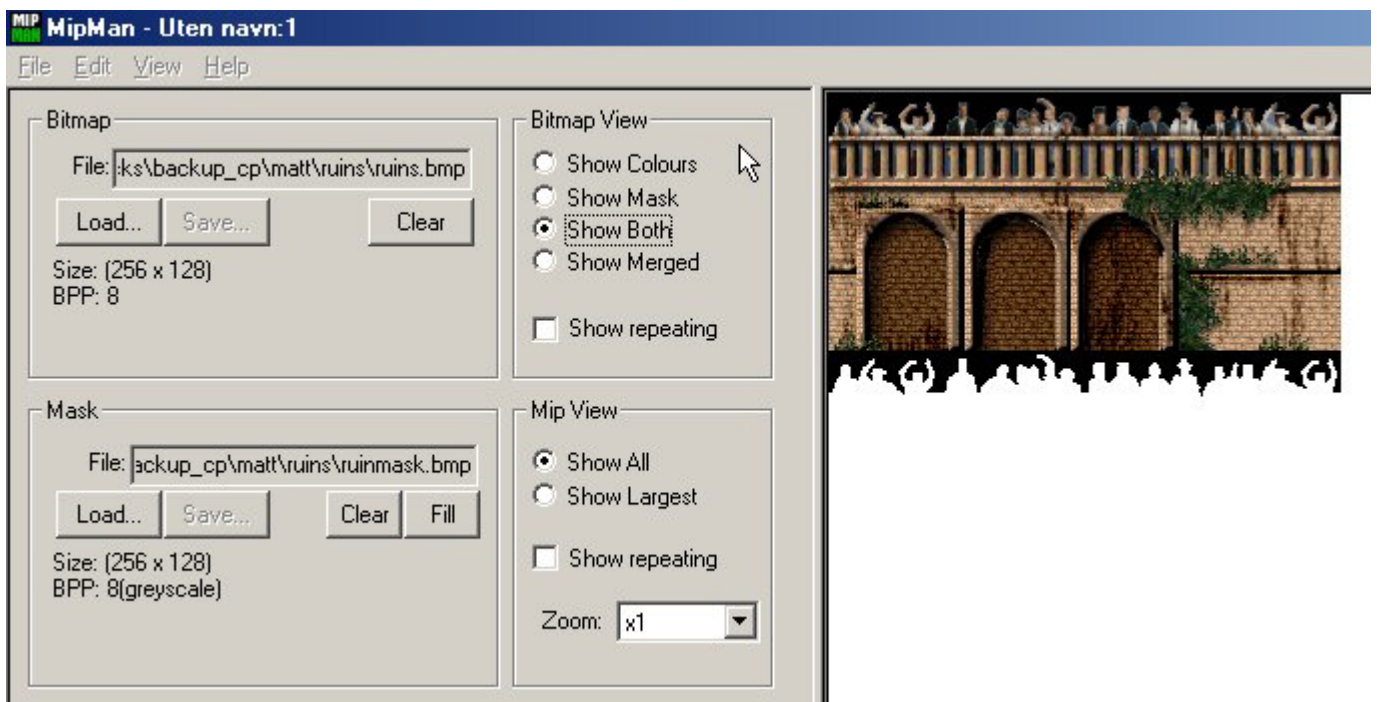
This mip was for the high-resolution version of Crystal Palace, so to make it look at all good, I needed to make it 16-bit, but I wanted to have the background behind the spectators on top transparent.

MipMan!

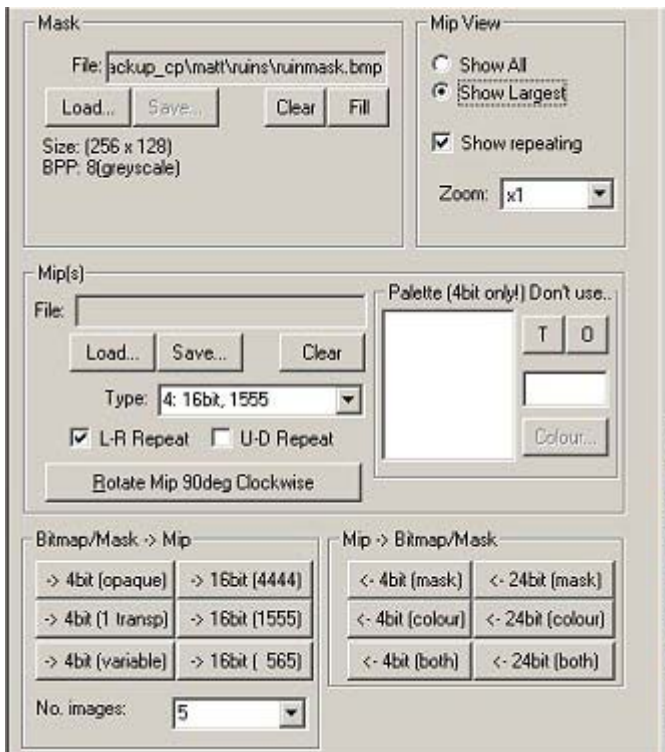
To make a the mip, get MipMan, a tool I rate as the nicest for making 16-bit ones. Here's the interface:



The interface may seem scary; but it isn't; there just is a lot of functionality! Start by loading your texture and mask. Then it should look like this:

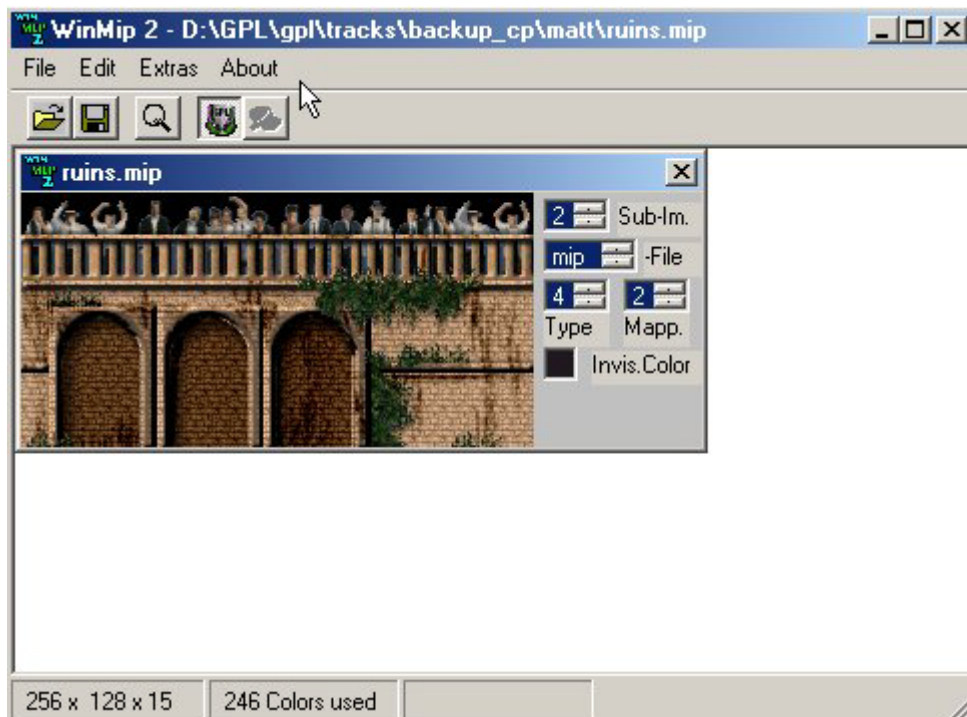


Here I've selected "show both" - this displays both the texture and mask, you have other choices too. But now we're ready to convert..



MipMan now reads the transparency from your mask, and by choosing type (16, bit, 1555), ticking off on L-R repeat (doh! Left-right repeat!). I now have the mip. There's one selection more you should mind: To the left, on the bottom: No. images: this sets how many sub-images GPL sees, or that are used in the mip. A higher number makes it look awesome from all distances & angles, but the mip becomes BIG and FPS Poor. So try and keep it low! But it has to be higher than 0!

To see what it looks like repeating: click on show all then show largest, then tick "Show repeating". You can mess about with all the settings and convert to your hearts content here, then remember to SAVE it with a proper name (not more than 7 characters by the way else you'll not be able to pack a DAT file later on. And presto! You have a mip! My preference is MipMan for making mips from scratch, if you already have the mip, modify the texture, and convert, Klaus Hörbrand's WinMip is the tool since it has a great batch function for converting from MIP to BMP and vice versa. So let's look at WinMip!

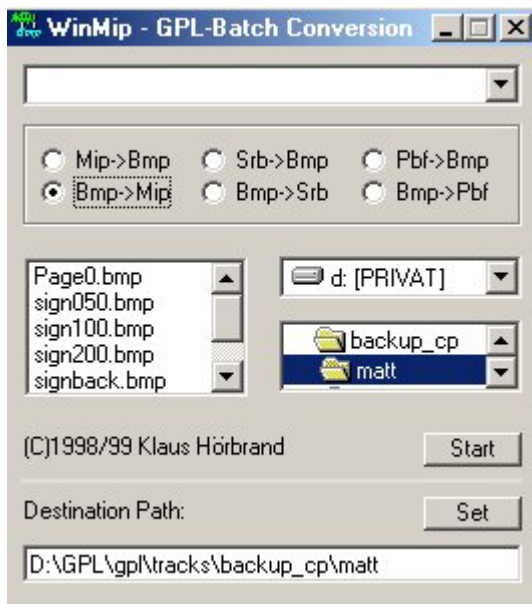


Here's my ruin opened in WinMip. The first law of WinMip: always have a mip with the same name as your BMP if you are converting BMP->MIP, WinMip reads info from the "old" mip when converting. You can of course cheat, use a mip you know is repeating, draw a new texture, give them the same name (new.bmp & new.mip), put them in the same directory, and convert. In this view you can change a few things, I suggest you load a mip, and change them and see what happens.....:)

TIP!:To make sure that your track will run in software mode: the field marked Sub-Im MUST NOT be 0!

If you're working in 4-bit, you can create a new Mip in MipMan in much the same way as I did the 16-bit one, drop the mask, and chose the 4-bit option on the converting buttons.

In WinMip, from the top menu, choose File->Batch Conversion. You'll get the Batch interface:



Here it is! Powerful! By ticking the radiobuttons you can choose different jobs, mip->bmp, bmp->mip. Converting both ways, and you select the mips/bmps in the browser window (you can select several at the time standard windows way by pressing CTRL, or bu click & drag). First you must find the right directory, then press the SET button. This will dump all results in the folder you have set. You can browse a folder, set it, and go convert it from a different folder if you so wish. I recommend keeping the mips/bmps in a separate working direcotry, and take backups along the way. If you've done something wrong, WinMip answers to this in a proper way; throws an error message (In german, and even though my german is bad, they always make more sense than the standard MS msgs!) and quits. WinMip is very nice for batch converting both ways, and it also has a neat-o feature under "extras" that unpacks/packs DAT files.

Beware: WinMip reads info from the old mip, and that includes all the properties. You can make a new size no problem, but WinMip will use the repeating value, number of colors and the transparent color from the old one. And don't try to convert using a 16-bit mip and a 4.bit bitmap :o)

More talk...

This should have been all the introduction you need. The appearance of textures are down to preferences in the end anyway. You also stumble into other files, called SRB. Making SRBs is very much like making mips, the same laws apply, and you can use WinMip or SRBMAN. But the SRBs have some functionality you should read up on in conjunction with 3dos. Basically, an SRB is a MIP that always looks straight at you, and is used for the flagmen and the animated objects. They also need a 3do.

So are there any secrets?

Nope, just work. When making textures, a digital camera is nice to have, some skills in Paint Shop Pro or Photoshop, tons of pictures, video, whatever you can get your hands on. The GPLEA have taken pride in thorough inspection of the tracks we have created (and it's fun to meet up with your netbuddies too!) And by that I mean pictures taken on-site used as textures as far as possible. You need to plan a bit, and work with the rest of the track; fake some shadows, use variations on themes (several types of grass) and so on. So get to work....

Trackside Objects

Adding to your track

There are 6 ways you can add objects to your track, all of which use the .TSO file. The first three are ways of placing the object within the driving limits of the track. The second three are the same, but involve placing the objects on the infield or outfield of the track. They are:

(1) TSO entry type 1 - absolute x,y,z coordinates, absolute x,y,z rotations. This is the least used method, because it requires the most maths. Look at the trk23dow.tso page for more information.

(2) TSO entry type 2 - longitude,latitude,z coordinates, absolute x,y,z rotations. This has been replaced by TSO entry type 3, the only difference is the absolute rotations, which I'm sure you can work out if you require this option.

(3) TSO entry type 3 - longitude,latitude,z coordinates, relative x,y,z rotations. This is the most used method of placing objects on your track. Here is an example TSO entry:

```
[ 11]: 3, 1839.4089, 0.0, -0.1016, 180.0, 0.0, 0.0, 1.0, mbridge comment
```

Hopefully you already understand what the numbers are, from the trk23dow.tso page. Here's a quick refresh anyway:

```
[ 11]:      a human-only number to identify this object
3:         TSO entry type 3
1839.4089  longitude - distance from start/finish line - in metres, of the 0,0,0 point of the object
0.0       latitude - distance from the centre line - in metres, of the 0,0,0 point of the object
-0.1016   z adjustment - distance from the track surface - in metres, of the 0,0,0 point of the object
180.0     xy rotation - rotation of the object in the top-down view
0.0       yz rotation - rarely used
0.0       xz rotation - rarely used
1.0       scale of the object - 99% of the time this is 1.0
mbridge   object name
comment   any old comment, for human purposes only
```

In this example, we have a bridge object being positioned so that it will fit entirely within one section (this is very important - see Avoiding Clipping for more details).

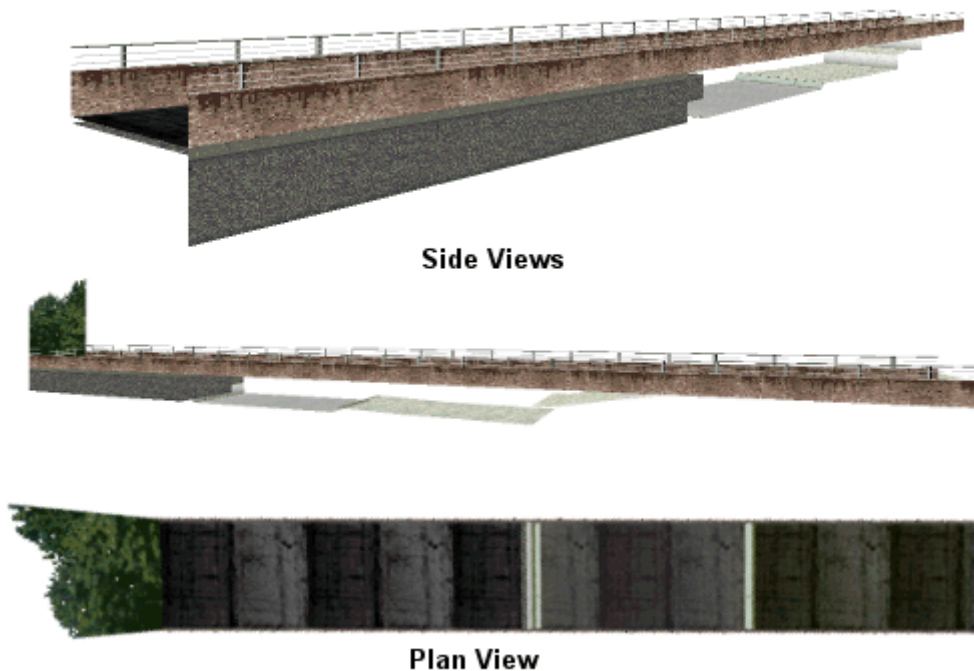


Figure 1. Here you can see how the bridge (the brown+railing parts only) fits in with the rest of the track. In Plan View, the bridge completely covers the track- if you were to see the next and previous track sections you would not see the bridge at all- this is A Good Thing.

Z Coordinate

You may be wondering why the Z coordinate of the object is negative ? This is because the object was designed to fit the track section, but it was designed for a flat piece of track. When the track was changed, the piece of track the bridge was designed for was made into a slope, so to make the bridge fit again, a slight adjustment was required

Infield/Outfield Object Placement

[Note 28-Jan-2001:] Infield/Outfield object placement seems to cause an abnormal amount of clipping. I'm not quite sure of the cause, although it sometimes helps if the outfield objects are listed as infield, and the infield as outfield (no, that isn't the error!).

Sometimes you don't want an object within the driving limits of the track- in fact, a great number of object in the original Papyrus tracks are outside the driving limits. One very good reason for placing objects on the infield/outfield is that the problem of clipping is greatly reduced, because there is no track to clip with. You can also have objects that are 'next to' several track sections, which you cannot have with the in-driving-area objects.

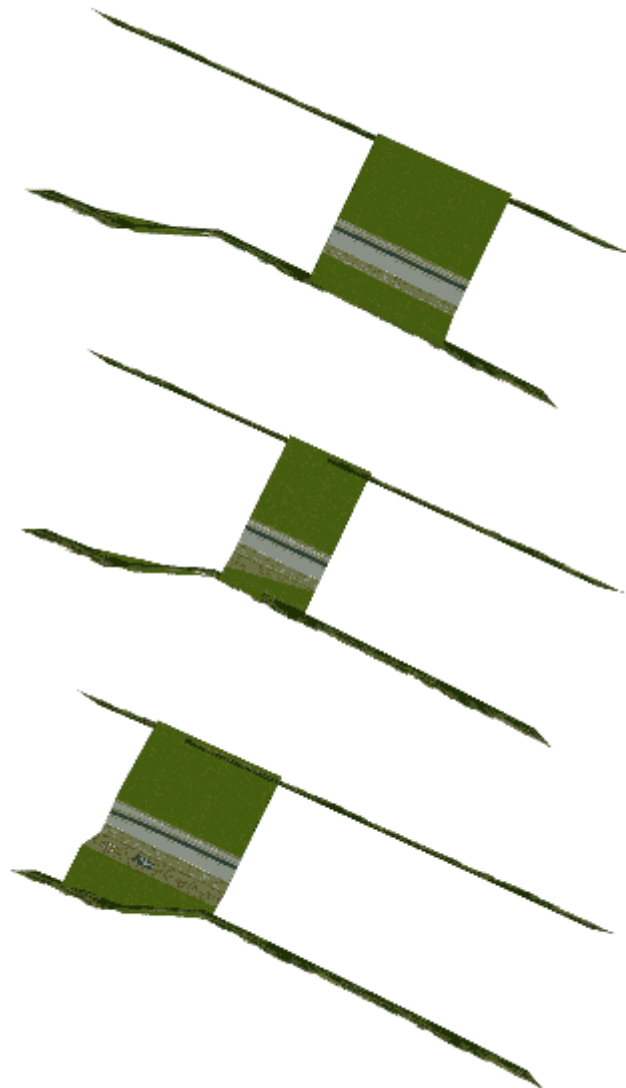


Figure 2. Here is an example from Rouen, where three consecutive track sections all reference the same two outfield objects - a row of trees on the left, and the right, of the track. In other words, the tree objects are 'next to' all three track sections.

To use the infield/outfield object placement, you must reference each track section that the object is next to, in the .TSO file. The `trk23dow.tso` page details this in more detail. You will, however, have to update these references if you change the track layout, so it is worth getting the track layout finished before you start with the infield/outfield objects.

Avoiding Clipping

Clipping is a very annoying problem. It is caused when polygons are drawn in the wrong order-overlapping each other incorrectly. Here is a cause-remedy table that should help:

Cause/Problem	Remedy
An object appears through another one - e.g. a row of houses all in one track section.	Change the order of the objects in the .TSO file so that the furthest away object (when driving normally) is listed first, the nearest object last.
Object appears to float/sink.	Check that the object is matched with ground-level. If it is, check that it lies completely within the track section (see bridge example). If it still clips with the track, make sure it lies completely within the left-right bounds. These bounds are defined by armco, walls, and the left/right edges of the driving area.
Several objects lie in a group near each other, clipping with each other from various angles.	If changing the .TSO order of the objects does not help, and you cannot move some objects to another track section, you will have to rebuild the objects as one 3DO. The buildings near the bridge at Silverstone are a good example of one object-many buildings.

Creating new objects

Creating new objects is a very time-consuming task. If you want to, [here](#) is a tutorial. You are also advised to read all the pages at [GPL File Formats](#). An understanding of vertices, planes, normals, texture mapping and Binary Space Partitioning is very useful as well.

Horizon

Sky and trees

You must have a `horiz.3do` for your track - this contains a sky 'box' that covers your track completely, providing the appearance of distant clouds & the afore-mentioned blue stuff. Unfortunately, `horiz.3do` is an evil object - it can be edited with `GPLTrackEditor`, but it is a nasty business, so we suggest 'borrowing' one from another track. Find a track that is similar to your own, and once you've made your own .MIPs for it's `horiz.3do` - containing the actual horizon, plus the sky - you should be okay. The Repository will be the place to find pre-made `horiz.3do+.mips`, once they've been made.

Does it have to be a sunny day ?

No way ! Jay Beckwith / GPL Team Friendship has shown that sunny isn't the only way with some wonderful overcast horizons, and in 5 minutes I turned Spa into night time, so use your imagination ! To be really corny: "the sky's the limit!" <insert silence>.

Program Covers

(PBF-files)

by Meik Thiemann

For displaying your track in the menus of GPL you need a set of graphicfiles (PBF-Format):

- BWPIC.PBF - A Photo for the Newspaper after a Race-event.
- EVENT.PBF - The Picture on the trackscreen, showing Infos and a trackmap
- PAGE0.PBF - Titlescreen for the trackselectionmenue
- PAGE1.PBF - Left Side Infos (trackselectionmenue)
- PAGE2.PBF - Big trackmap (trackselectionmenue)
- PAGE3.PBF - Backgroundsheet for Championship-standings
- PAGE4.PBF - Backgroundsheet for Championship-standings
- PAGE5.PBF - Backgroundsheet for Championship-standings

You have to use a graphicprogram of your choice to paint these files and convert them to PBF-format.

Tools needed:

GPLBatch by Klaus Hörbrand

GPL Replay-Analyser by Jonas Matton and Martin Granberg

dooDAT by Double D Software

A graphicprogram of your choice ...

Files needed:

- Futura Light Condensed BT - Font (can be found on CORELDraw-CDs)
- UNLICSND.PBF
- OPENRIT.PBF
- OPENLFT.PBF

These images can be taken from the LAYOUT.DAT of GPL. Extract them using dooDAT and convert them to BMP.

BWPIC.PBF:

Dimensions: 275x209Pixel

To get a nice picture for the newspaper, take a screenshot of some cars in a race. In your graphicprogram take out a fitting section and scale it to 275x209 Pixel. Reduce it to greyscale and back to RGB. (Converting it back to RGB prevents conversion-problems in GPLBatch). Save it as BMP-File.

EVENT.PBF

Dimensions: 258x340Pixel

This page contains 4 elements: A flag of the country, the name of the event, the name of the track and a trackmap. Some other information like number of laps, tracklength and record is automatically added by GPL itself. Make sure not to overpaint these used sections.

Use UNLICSND as Background for your EVENT-page.

How to get a Flag: Try to extract it from existing GPL-tracks, otherwise you have to make one yourself. Maybe you have some graphic-collections like CORELDraw or something similar. Sometimes you'll find collections of countryflags there. The size of the Flag is 58x41 with a 1 pixel wide, black border.

The Name of the event is written using the 'Futura Light Condensed BT'-Font. For the textposition- and heights you should take a look at the existing tracks. The name of the track is written using Arial-Font, wich is a standard Windows-font.

On how to make a trackmap, see the separate section below. For the position and size of the trackmap take a look an the original tracks. Note that the trackmap here contains all details of the

bigger PAGE2.PBF except the Pits.
Save it as BMP-File.

PAGE0.PBF

Dimensions: 281x431

This is the figurehead of the track. Now you have to use your creativity to make a unique design. Some infos like eventname, trackname and date should be included here. As a little help I would recommend to take a look at The Programme Covers Project of Malcolm Mitchell <http://www.malcolm.mitchell.btinternet.co.uk/>. Here you'll find a covers of a lot Motorsportevents. You only should look at the 1960's covers to fit the right design. After you have painted this, you should overlay it with OPENRIT. Use an overlayfilter like structure or multiply. This gives your page the look of a booklet.
Save it as BMP-File.

PAGE1.PBF

Dimensions: 281x431

This page contains 3 elements: the flag of the country, the name of the event and the name of the track. Like in EVENT.PBF there some infos added by GPL to this page (number of laps, tracklength, record). Make sure not to overpaint these used sections. Take OPENLFT as backgroundimage. The font for the eventname is 'Futura light condensed BT', the font of the trackname is Arial again. Note that the size of the flag here is 57x39 (in EVENT.PBF it is 58x41!) for the position and size of all these elements take a look an the original tracks.
Save it as BMP-File.

PAGE2.PBF

Dimensions: 281x431

This Page shows a big trackmap with all details. It contains the sectionnames, the north pointer, position of start/finnish-line, the position of the Pits and the direction pointer. The trackmap should fit the complete page (of course with some border). All elements like the direction pointer, or the north pointer can be taken from the original tracks. Use OPENRIT as backgroundimage.
Save it as BMP-File.

PAGE3.PBF - PAGE5.PBF

These files are used for the championship-standings. You only have to include them if you want to add your track to the GP.INI. They don't need to be modified, take them from an original track.

How to make a trackmap:

To make a trackmap, you have to use GPL Replay-Analyser. Go to your track and drive a last one timed lap. Save the replay and open it in GPLRA. You should see your trackmap now. Go to File/Settings and change 'Size of exported BMP file (Pixel)' to a minimum of 3000. Use File/Save trackmap as BMP-File to get a bitmap of the track. Now there are two ways to get it into your program cover.

The easy way:

Load the bitmap into your graphic program and remove all colours except for the asphalt. Reduce it to 1bit colour and add all other things like sectionnames, north pointer, etc. Before you can insert the trackmap into EVENT.PBF or PAGE2.PBF, you have to rotate it (horizontal for EVENT.PBF, vertical for PAGE2.PBF).

The difficult way:

For this way you must be familiar in painting with vectorgraphics. Import the bitmap into your program (I prefer Micrografx Designer) and redraw the track by painting over the bitmap with lines. To you have your track as a vectorgraphic makes it possible to rotate and scale it as you want (without reducing the quality). And it is easier to adjust possible faults. When you think everything is fine, convert it to bitmap and use it in EVENT.PBF and PAGE2.PBF.

Converting:

After you have made all this pages and saved them as BMPs, use GPLBatch to convert them to PBF and include them into your TRACK.DAT. If you get any problems with purple-graphics, make sure that the BMP-files are 24-bit clour before converting to PBF.

Still confused?

Still confused, or your work isn't satisfying? I'd like to offer my help for all the trackbuilders: If you have any question regarding on program covers, send me an email. Maybe I have the time to make all this work for your track: meik.thiemann@gmx.de

AI by Ed Solheim

To make AI-files for GPL you need this guide and 7 other things:

- 1) A "clean" driveable track.
- 2) RPY2LP by Nigel Pattinson [available at www.theuspits.com]
- 3) LP-Edit by Frank Dubuc [available in [downloads section](#)]
- 4) The TRACK.INI GUIDE by yours truly.
- 5) A spreadsheet program such as Excel.
- 6) A good, clean and preferably fast driver.
- 7) Last but not least - plenty of time!

Track

IMPORTANT: Before you begin creating AI's for any track that has been created with *TRK23DOW*, you need to prepare the TRK-file so RPY2LP can read it (see bottom of guide for clues as to what happens if you don't prepare the TRK-file correctly. As you might know, *GPLTRK* and *TRK23DOW* use the UNK 3's and 4's in the TRK(GTK)-files for building 3do's. Unfortunately, RPY2LP does not like this information at all - thus one has to 'clean' a TRK-first, before converting the replays to LP-files. Cleaning these files can be done using *TRK23DOW*; Just save the GTK-file and run it thru *TRK23DOW* using the z-syntax (i.e `trk23dow -z trackname`). This will then create a GT2-file (trackname.GT2) - rename this GT2-file to .GTK and load it into *GPLTRK*. If you look at it in *GPLTRK* you will notice that all unknown 3's and 4's have been removed. The last step now is to export the file as TRK and put it into your track-directory (i.e `..\gpltracks\trackname\`) **before** you try and convert an RPY to LP with *RPY2LP*. If *RPY2LP* still gives you problems.. you will need to edit the "cleaned" GTK-file accordingly - see bottom of this document for more info.

It's important to notice that the number of records in the LP-files is dependent upon the length of the track. Should the number of records in the LP-files be incorrect, GPL will simply ignore the files completely and you'll end up with the AI's "coasting" around the track in the middle of the road at 40mph or so...

GPL uses 1 record per "tick" [1 "tick" = 1/36th of a second], so if you change the track-length you're screwed and have to start making these files all over again. This is why I always wait until the last possible minute [prior to release] before I make the LP-files for the AI's.

Another thing to point out is that everything that will have an influence on laptimes or speed should be finished before you go and make these files. Things such as elevations, track "camber", surface-types etc. all determine what sort of pace you can keep at any given point on the track, and small changes in these areas can have a great influence on speed - so you should wait until these bits are "set in stone" before you go on to make the [final] version of the AI's. I say 'final' because - it doesn't hurt to practice :)

LP Files

The LP-files you see in each track-directory are used to control the AI's behavior. GPL use 6 such files:

Race.LP:

This file controls the speed and line the AI's use for "normal" driving/lapping ["alone"] around the track.

Pass1.LP:

Controls the AI's line and speed when passing [and possible being passed (on the right-side)] a car on the *left* side of the track.

Pass2.LP:

Controls the AI's line and speed when passing [and possible being passed (on the left-side)] a car on the *right* side of the track.

Maxrace.LP:

Defines the left-most "edge" of the track. Used by the AI's to determine where they can drive if the road ahead is blocked etc. Also for SHIFT-R resetting (more on this later).

Minrace.LP:

Defines the right-most "edge" of the track. Used by the AI's to determine where they can drive if the road ahead is blocked etc. Also for SHIFT-R resetting (more on this later).

Pit.LP:

Controls the line and speed AI's use when exiting and entering the pits.

File Structure

Each LP-file contains X number of records [one for each "tick"] and each record has 5 fields of data. These are [in correct order]:

- 1) Longitudinal speed
- 2) Lateral Speed
- 3) Lateral Position
- 4) Yaw Velocity
- 5) Waypoints (or "flags")

If you convert an LP-file to text and open it in Notepad, the records will look like this:

1.9253, 0.0571, 0.1838, 0.0002, 16

This record shows that:

- the car's speed is 154.024 mph. (80×1.9253 (value = meters per tick))
- lateral speed is 0.0571 meters per tick (could be 4.568mph (i.e 80×0.0571))
- the car's [center] position is 0.1838m to the left of the track center-line (positive value is always on the left-side)
- yaw-velocity = 0.0002 radians per 1/36th of a second or tick.
- the waypoint or flag to tell the AI that a long straight is ahead.

Way Points or Flags

Meaning	Hexadecimal	Decimal
Passing Checkpoint	0x01	1
Set Checkpoint Speed	0x02	2
Outbrake Decision Point	0x04	4
Clear Passeur	0x08	8
Start Long Straight	0x10	16
End Long Straight	0x20	32

Start No Pass Zone	0x200	512
Clear No Pass Zone	0x400	1024
Tire Warmup	0x10000	65536
Speed Modifier	0xFF000000	big !

A quick run down of the various flags:

PASSING CHECKPOINT:

Always seems to be followed by another flag, so I assume this tells the AI to prepare for an upcoming flag. Could also be that this in fact tells the AI's to "pass here".

SET CHECKPOINT SPEED:

I assume this is used by the AI's to check their pace/speed compared to the information in the LP-file.

OUTBRAKE DECISION POINT:

Used to define outbraking zone.

CLEAR PASSEE:

Normally used after (or just inside) a NO PASS ZONE.

START LONG STRAIGHT:

Tell the AI's that a long straight is ahead. Probably to encourage slipstreaming.

END LONG STRAIGHT:

Used at the end of straights - see above.

START NO PASS ZONE:

Self-explaining.

CLEAR NO PASS ZONE:

Ditto.

TIRE WARMUP:

Due to the fact that the AI's in GPL use a much simpler physics engine than the players, the AI's do not actually generate heat in their tires. Thus the need for a flag to increase the heat in their tires. This flag is normally deployed every 400 records (depending upon the track) or so. When the AI pass this flag - its tires will be heated up to the next level. This continues until the tires reach their optimum temperature - after which the flags will be ignored.

SPEED MODIFIER:

The SPEED MODIFIER is used to speed up the AI's at specific places around the track, probably at places where they are too slow etc. So if you make some AI's and they turn out to be a bit slow at certain spots you can use this modifier to speed them up.

The modifier is scaleable - i.e. you use the two topmost bits to adjust how much faster you want to make them. So, one speed modifier can look like this: 0x02000000 (this only speed them up a "tad") or you can make one that looks like this: 0xDF000000 which make them go a lot faster. The MODIFIER must be present in all the records you want to "speed up" (look at the Kyalami AI-files and you'll get the idea).

The SPEED MODIFIER can also be combined with any of the other flags - like this: 0x3C010000 (MODIFIER and TIRE WARMUP) or 0x08000400 (MODIFIER + CLEAR NO PASS ZONE)

PLEASE NOTE: The MINRACE, MAXRACE and PIT.LP files **DO NOT NEED ANY WAYPOINTS OR FLAGS IN THEM AT ALL!!**

How to make LP-files:

You convert replays to LP-files with RPY2LP. Note that each lap must be a complete lap - i.e. you must pass the start/finish line *twice*. To give you an idea.. this is how I make them:

MAXRACE.LP:

Use one of the widest cars in GPL (Lotus or Eagle). Drive a complete lap of the track along the left-side of the asphalt. I also move a little away from the track-edge when driving along the outside of a corner. Speed is 30-35mph as this is the speed the AI's use when they "limp" back to the pits after a breakdown etc.

MINRACE.LP:

As MAXRACE, only that this time it's along the right-side of the track.

PIT.LP:

This lap has to start and end in the pits.. I do NOT suggest that you stop in the pits or just start from the pits when doing this lap. Do one lap first... then drive into the pits and through it at 10-15mph, then exit the pits like one normally does. Drive around the track and then enter the pits again, like if you were pitting. NOTE: all of this depends upon *where* the pits are situated in relation to the S/F-line.

RACE.LP:

This is the file used for "normal" driving/racing. Make this out of a normal "hotlap" - where you drive the line and speed you normally would in a race.

PASS1.LP and PASS2.LP:

Now this is where it's getting tricky. The quality of the AI's ability to race both humans and each other, depends upon the "quality" of these files. It might look like there are various ways to make these.. if you look at the original tracks, you'll see that on some tracks the passing line is made by just driving on either side of the normal line throughout a whole lap. And on a few other tracks the Lateral-position in these files will revert back to the normal [race.LP] position after a "passing move".

Drive a complete lap of the track at race-pace using the line you would for passing on the left-side [for pass1.LP] and one lap using the line you would for passing on the right-hand side for the pass2.LP.

Getting It All To Work

Now that you've made all the replays for the LP-files.. you need to convert them using *RPY2LP*. Once you have them converted, either use *LP-Edit* to add flags etc. or better yet, use *LP-Edit* to convert the LP-files to Text, and then use Excel etc. for the flags.

Please note that in *LP-Edit* you need to enter the waypoints/flags as Hexadecimal. If you convert the files to Text and use Excel etc., you will need to add the flags using Decimal formats. If you want to add some speed-modifiers into the files, I strongly recommend that you use *LP-Edit* for this task as adding them as Hex is much easier than Decimals.

Also, when exporting or saving the files from Excel, they must be saved in a *comma-separated* text-format for *LP-Edit* to import them properly.

Now all you need is plenty of spare time to get these damn things to work!

More Helpful Hints

When creating AI's for "add-on tracks" you might run into a few problems:

On some tracks you'll see that the AI's you create seem to drive badly through chicanes and esses etc. Sometimes they will also seem to be somewhat "out of sync" at the latter parts of the lap. I've found that on most tracks, and especially those where *RPY2LP* has to "estimate" the number of records needed for the files, the first record in the LP-file will be based upon data from what really should have been the last record in the file. You can easily spot this by comparing the data in record #1 and #2 or #1 and the last record in the file. This actually causes the AI's to be somewhat "delayed" in their behavior, and this problem seems to grow the further into the lap they get. Moving the first record to the end of the file seems to fix this.

Another thing worth mentioning is that you should make all the LP-files [based on correct replays] before you start to adjust or edit them. Sometimes I've been working on small problems for hours, only to find them being fixed by adding correct pass1/2 - files.

Sometimes you might see the AI's lift or slowdown slightly at different spots.. this is normally an indication the the race.lp line crosses either the max- or minrace.lp lines. To correct this problem, adjust or re-do (read: drive a tighter/wider line) the files causing the problem (most likely the race.lp file)

I also strongly suggest that you use *LP-Edit* to explore the original LP-files that came with GPL. There are plenty of things to learn from them.

I also suggest you get hold of my TRACK.INI SETTING GUIDE, as this will give you all the information you need for the TRACK.INI.

Help, RPY2LP won't let me convert the replays!

"GPL trk file seventh surface value not zero":

TRK-file still contains Unknown 3's and 4's.. See top of file for "cleaning-instructions"

"GPL trk file section start error":

Only applies to add-on tracks AFAIK. This is caused by a start mismatch in one or more of the track sections. To fix this problem, open the "cleaned" GTK-file in *GPLTRK* and look thru all the sections for a start mismatch. If one is present you will see it clearly! Should no mis-matches be present then look at the first section of the file (Section 0) - it will probably have a longitudinal gap. To get rid of this gap, increase or decrease the *length* of the **last** track-section.

Happy editing!!

.ini Files

track.ini is first explained - the file that controls race behaviour and many important bits of your track.
67season.ini / gp.ini are second - how your track is found by GPL, VROC, etc.

Creating a DAT

GPLDFM

This is a simple little MS-DOS program that will create a DAT file for you. To use it, copy it to your track directory (sierra/gpl/tracks/trackname). With that directory open in Windows, do Start->Run->command. Now, type:

```
GPLDFM -c trackname.dat directory_name
```

And all files from directory_name (e.g. c:\sierra\gpl\tracks\mytrack) to the given DAT (trackname.dat). Files added are: (*.mip, *.3do, *.pbf, *.srb, *.cam, *.trk, setup.*).

If you already have a DAT, it will be overwritten. Run GPLDFM without any parameters to get a list of the options (not all of them work, hehe).

Other Programs

There are many other programs that will create DAT files for you. Have a browse around and find one you like!

Frequently Asked Questions

This page should be your first port of call if you have a question. Please take the time to look through these FAQs. If you do not find an answer, please visit the **forum**, **do not email us - we will not reply to any emails regarding track creating/editing.**

Sections

[GPLTrk; TRKs; GTKs](#)
[trk23dow; TEX; TSO; GRV; FB](#)
[AI; .ini Files; track.ini; 67season.ini; gp.ini](#)
[Images; MIPs; SRBs; PBFs; Program Covers; MipMan; WinMip; SrbMan](#)
[Miscellaneous](#)

GPLTrk; TRKs; GTKs

GTK ? TRK ? Huh ?	A TRK file is the physical layout of a track, required by GPL. A GTK file is exactly the same, except GPLTrk will only Load GTKs, not TRKs. This is to prevent people editing tracks to gain an advantage (i.e. cheating). GPLTrk will convert a GTK into a TRK for GPL. trk23dow will only read GTKs
GPL crashes/freezes when I cross from one section to another	This is caused by gaps/mismatches in the TRK sections. When there is a gap, trk23dow will not fix it, and will create a track with empty spaces - when you drive on those spaces you are driving on 'nothing', so GPL cannot work properly. See the Creating a TRK.Sections page .
I've made the whole track but can't get the last section to meet the first.	1) Fix any Orientation mismatch. 2) Fix lateral gap by changing corner radii. 3) Fix longitudinal gap by changing the length of a straight. See the Creating a TRK.Sections page .

[back to the top](#)

trk23dow; TEX; TSO; GRV; FB
errors, output 3DO problems

My track has bad FPS / jerky motion	See the FPS page.
Can't see enough of the track ahead / objects pop into view	You need to edit the <trackname>.FB file. This file controls the distance ahead and behind that you can see whilst driving/watching a replay. You should keep the numbers in this file as small as you can, to avoid overloading the CPU with unnecessary info. Look at the .FB guide for more detail.
Track is visible through objects	This is clipping. See below.
Track is visible through the track - where there are hills	If you can see parts of the track where you shouldn't be able to - e.g. you are behind a bit of banking and can see the track - then you need to experiment with unk4 flag 1024 . This flag is used to split the world into chunks - you need to split the world where the banking meets ground-level. If you can see other weirdness that isn't solved by this problem, please visit the forum and ask there.
My object doesn't show in GPL	Check it is in the .TSO file (properly!) Filenames beginning with numbers, and/or containing symbols are not valid. Check that the object is actually positioned within the driving limits (if it is a 'normal' TSO entry); completely on the infield (if it is an infield TSO entry); completely on the outfield (if it is an outfield TSO entry). If you do not make sure of this, the object will probably be ignored by trk23dow. Check that the object isn't being hidden by another object, and that the object is at track level. Check you haven't placed the object on e.g. a piece of armco. trk23dow will make the object visible on the armco, but because that is a thin piece of the 'world', it will be clipped and lost.
An object is clipping with another object	Re-organise the objects in the .TSO file. For each TRK section, you can have several trackside objects-- you should have the furthest away object in each section listed first in the TSO file, working back to the nearest object. (This is when you are driving forwards).
The groove is too blocky / erratic	A blocky groove is caused by not enough polygons. Use the unk4 flag 8192 to increase the number of polygons. An erratic groove is caused by a poor replay - see the GRV tutorial for information.
The cars float above / sink below the track surface	This is caused by not enough polygons representing the track surface. Use of unk4 flags 2048 and 4096 will increase the number of polygons for individual TRK sections. Use of the <trackname>.SET file will increase the number of polygons/sections for an entire track.
trk23dow loops when working with the TSO file	The cause of this is a faulty .TSO file. Check that you do not have any blank lines, that the number of entries values are correct, that each line has the correct format. Also, make sure that no 3DO files begin with a number, because this sometimes confuses the program.
The track is bumpy	Use either gtk2csv to edit the altitudes manually to smooth the angle changes, or use trk23dow -f or -s to smooth the track automatically.
When crossing sections, I get a spike in my FF wheel	Caused by TRK sections overlapping. ->

[back to the top](#)

AI; .ini Files; track.ini; 67season.ini; gp.ini

Could Not Load Track - track name isn't shown in the program cover	track.ini is missing or invalid.
Track isn't shown in the selection screen	67season.ini / gp.ini is invalid. Check the number of events value in each .ini, and make sure that the entry for your track is correct. Also, if an event is repeated, GPL sometimes goes a bit wonky.

[back to the top](#)

Images; MIPs; SRBs; PBFs; Program Covers; MipMan; WinMip; SrbMan

My track doesn't work in software rendering	This is caused by bad .MIP files. There are many ways a .MIP can be bad, yet still work in hardware rendering. Load your track in RaceCon, turn on all the Texture options and View->3D. RaceCon will tell you if there is a problem, and with the treeview you can narrow down where the problem is. If you find a duff MIP using RaceCon, load & save it in MipMan (no editing required) to fix it.
Some textures are stretched like thin lines	Use MipMan/WinMip to set the repeating values, so that GPL will tile the textures instead of stretching them (badly).
The track asphalt looks generally weird	Use <u>unk4 values 2 or 512</u> to make the track tiled once left-right. 2 is used when the track camber is linear, 512 is used when the track camber is curved.
I can only see the track; my walls are missing	trk23dow will clip everything that is not within the range of the traces. Use GPLTrk to add/move the traces further away so that all walls are within the traces.

File Types

<u>DAT files</u> : Wrapper	These are a wrapper for the hundreds of files that make up a track (or any other resource in the game)
<u>3DO files</u> : 3D objects	Defines the 3D objects, e.g. cars, track, trackside objects...
<u>MIP files</u> : "Multum in parvo" (thanks Oliver)	Store textures in several resolutions
<u>SRB files</u> : Single/Scaled Resolution Bitmap?	Also textures, but in a scaled format? Contains within a Mip file.
<u>PBF files</u> : Pictures	Pictures for the non-game aspects, e.g. track posters.
<u>TRK files</u> : Track layout (by Nigel Pattinson)	Stores the information required to drive the track,

<u>TRK Tutorial</u>	not see it. Also, a tutorial
LP files: AI driving lines	So the AI cars can drive <i>perfect</i> laps, overtake, etc.
<u>CAM files</u> : Cameras	Camera locations for TV1, TV2 & Pitlane.
<u>CCAM files</u> : Car Cameras	Car camera files
<u>SETUP files</u> : Car setups	Detailed setups, as changed from the car setup screen.
WAV files: Wave files (non GPL specific)	Sounds, believe it or not...
<u>controls.cfg</u> : Controller config	Contains the controller setup info. (1 per player)

Grand Prix Legends .DAT File Format by P.A.Flack 1998

last revised 14/9/1998

The .DAT files are used to group together many files into one. The structure of the file is fairly straightforward as it consists of only three parts - a header; a directory; the files. The header looks like this:

Bytes	Values	Notes
0-1	no. entries	Number of directory entries

Each directory entry looks like this:

Bytes	Values	Notes
0-1	0x05 0x00	always 0x05 0x00 for whatever reason
2-5	file size	the size of the file, in 4byte, lsb first
6-9	file size	the same (why???)
10-22	file name	name of the file, padded with 0x00's
23-26	file start	the starting position of the file within the DAT file

Then the files follow, consecutively.

Grand Prix Legends .3DO File Format by P.A.Flack 1999

>> **INCOMPLETE!** (but getting there...) <<

last revised 7/June1999



07/June/99: More information on the Tracks section (almost finished !)


24/May/99: Complete reorganisation; new animation :-)

The 3DO files are the most complex I've seen as they consist of several sections which are linked together. At the start of a 3DO file is a header of the standard sort (see the [main page](#)). This header serves simply to show how big the rest of the file is.

The various sections that make up the 3DO file are now listed, but not in the order they appear in the file. Not all sections always appear either, and the header doesn't seem to offer any clue

as to which sections do exist. The only way to find out is to read each section in turn, and find out what sort it is by looking at the section header (standard sort).

The structure of this page has been reorganised as follows:

 Vertices, Planes, Normals and Strings

.3DO File Format - Vertices, etc. by P.A.Flack 1999

last revised 24/May/1999

XYZS - Vertices

This section gives the vertices for the polygons which make up the object(s). The numbers for the vertices are 32-bit floats.

Bytes	Values	Notes
0-3	"SZYX"	Section Type (reversed)
4-7	0x00000000	Seems to be a zero all the time
8-11	data size	Size of data in bytes
12-	data...	Sets of 16 bytes

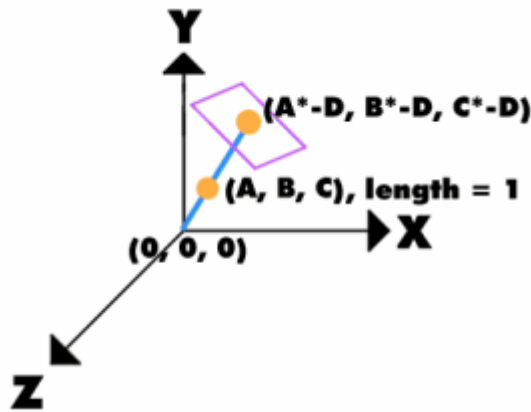
The data section consists of multiples of 16bytes: 4bytes each for the s, x, y & z coordinates (in that order). The s coord has been zero in all the files I've seen. The coordinates are in float format, and can be read (in C) with:

```
float x;  
FILE* f;  
fread(&x, sizeof(float), 1, f);
```

PLAN - Planes

This section looks very much like the Vertices section, but the use of the values is different. The header is the same as for XYZS, apart from bytes 0-3, which are "NALP". The contents are used to define planes, as in $Ax + By + Cz + D = 0$. The length of the vector A,B,C is 1.

Here is a diagram to hopefully show how these numbers define a plane:



The point (A^*-D, B^*-D, C^*-D) can be thought of as the centre of the plane. In reality, planes stretch off to infinity and there is no centre. Planes are oriented at right angles to the line shown.

Planes are used in GPL for two purposes - for Hidden Surface Removal (HSR), and Collision Detection. In both cases they are used to split 3D space into two parts, *above* and *below*. GPL also uses *on the plane*, and sometimes uses - *on the plane viewed from above*, and *on the plane viewed from below*.

↑ NORM - Normals

I've seen this section in the car 3DO files, and I'm guessing it's used for lighting effects. The layout is fairly similar to XYZS & PLAN. First value is homogeneous coordinate, next three are x,y,z.

In general terms, normals are unit length (1.0) vectors that define the direction a polygon lies in - e.g. for a table top the normal would point straight up. Normals can be defined either once for the entire polygon (used for polygons that represent flat objects), or once per vertex (used for spheres, etc.).

▲ STRN - String Table

This lists strings which give the file names used by this object. No file extensions are given - the extension is implied in the PRIM section (see later). Each string is terminated with a 0x00 byte, and the STRN section is terminated with 0xFF. Not included in the size of the section (taken from bytes 8-11 of the header) is the number of bytes required to round the size of the STRN section to a 4 byte multiple. 0x20's are used for this purpose (spaces).

A typical STRN section looks like this (kyalami/in-t9.3do, if you must know).

Bytes	Values	Notes
0-3	"NRTS"	Section Type
4-7	0x00000000	Seems to be a zero all the time
8-11	data size	Size of data in bytes (not including padding spaces)
12-	grass02, 0x00	Strings, each terminated with a 0x00 character
	0xFF	A single 0xFF character terminating the string table
		Padding 0x20 characters (spaces). Either 0, 1, 2 or 3, to bring the STRN size up to a multiple of 4.

3DO File Format - Primitives

by P.A.Flack 1999

last revised 7/June/1999

003 004 005 006007 008 00900A 00B 00D00E 00F 010011 013 014016 017 019A03
01F 020 021 815 81B81C 81D 81E81F 820 821

This part of the 3DO file is the most complicated and consists of many different runs of data which shall be referred to as Types, or T-xxx (hexadecimal). The header of this section is the same as all the others:

Bytes	Values	Notes
0-3	"MIRP"	Section Type (reversed)
4-7	0x00000000	Seems to be a zero all the time
8-11	data size	Size of data in bytes
12-	data...	Data, grouped into Types

The first four bytes of the data section (e.g. bytes 12-15 in the table above) give the starting point of the first Type *in the data section*. I.e. if bytes 12-15 give the number 5,000, this would equate to byte 5,012 including the header.

From this point on, the 32-bit numbers will be referred to in the 'correct' order. I.e. not like the "MIRP" in the above table. If you're looking at a 3DO file with a hex editor, the 4 bytes for each number will be in the 'incorrect', or network order: b3 b2 b1 b0.

Some numbers are 32bit integers, others are 32bit floats. The floats are generally used for coordinates (vertex & texture), and can be identified by their huge values if you read them in as an integer :-)

Most of the Types point to other Types. Those that don't are the T-8XX's which define polygons and a couple of others. Basically, the PRIM structure can be drawn as a tree. (Sometimes a recursive tree - track.3do's).

Some more notes:

- Colour means a 4 byte number in the format Alpha Red Green Blue ('correct' order).
- Offsets are offsets within the data section, i.e. not including the header.
- All numbers are given in hexadecimal, without the 0x prefix, but sometimes with, so beware :)
- Some offsets are 0xFFFFFFFF. As far as I can tell, this means that it should be ignored.
- Also, some vertices/planes, etc. are 0xFFFFFFFF. These should probably be ignored.
- String offset means an offset into the STRN section, not including the STRN header.
- Vertex numbers are not offsets into the VERT section. To get the offset into the VERT *data* section, multiple the Vertex number by 16.

- The same goes for vertex normals within the NORM section (if there is one in the 3do) & planes within the PLAN section.

Notation Used

These images mean: general node; leaf node (primitive or other); car-specific; finished!



General format is T-x (where x is hexadecimal); brief description; list of components that make up the type (nearly all 32bit integers/floats, where entries end with ..., it means there are a variable number of them); Long description

For the quick look table below, these images signify the different components:

=offset (i.e. pointer to a child node)

=colour

=string offset

=scale (float)

123=integer

6.8=float

=vertex number

=plane number

=normal number

= texture x,y pairing (2 floats)










































































number = this number appears















count = there are this many 'things' to follow - where a thing is in brackets and followed by ...

Quick Reference Table

(for you editor people with a busy life...)





Type (hex)	Format	Description
003	3, , , , , 123	SRB Chooser & Placer & Scaler
004	4,count, 123 's...	Multi child node
005-1	5, , 1,	Selector: mip file without display class
005-4	5, , 4, 123	Selector: ? display class ?
005-5	5, , 5, , 123	Selector: mip file with display class
005-8	5, , 8, 123 , 123 , 6.8	Selector: ?
005-C	5, , C, 123 , 123 , 123 , 6.8	Selector: ?

005-D	5,  , D,  , 123, 123, 6.8	Selector: ?
005-11	5,  , 11, count,  , 's...	Selector: ?
005-20	5,  , 20, 123	Selector: ?
005-24	5,  , 24, 123, 123	Selector: ?
005-2C	5,  , 2C, 123, 123, 123, 6.8	Selector: ?
006	6,  , 	Plane, single child node
007	7,  ,  , 	Plane, double child node
008	8,  ,  ,  , 	Plane, quad child node
009	9,  ,  ,  , 	Plane, triple child node
00A	A,  ,  , 	Plane, triple child node
00B	B,  , 	Plane, double child node
00D	D, 6.8 (dx), 6.8 (dy), 6.8 (dz), 6.8, 6.8, 6.8,  , 	Positioner
00E	E,  , 0	3DO File Chooser
00F	F,  ,  ,  ,  ,  ,  , -1, 0,0,0,0, count, (6.8, 6.8, 6.8, )'s...	Track visibility box
010	10, count, 123 's...	?Chooser of some description?
011	11, 123, 123, 123, count, (6.8, )'s...	Trk relation?
013	13, 6.8 (dx), 6.8 (dy), 6.8 (dz), 6.8, 6.8, 6.8,  , 	Positioner (3DO)
014	14, 123, 123	?
016	16, 6.8 (dx), 6.8 (dy), 6.8 (dz), 6.8, 6.8, 6.8,  , 	Positioner
017	17, count,  , 's...	Room?
019	19,  ,  ,  ,  ,  ,  ,  ,  , 	Bounding cuboid - seen lots of places
01F	1F, 0, 123, count,  , 's...,  , 's...	Car Polygon, textured
020	20, 0, 0, 123, count,  , 's...,  , 's...,  , 's...	Car Polygon, textured, smooth shading
021	21, 0, 123, count,  , 's...,  , 's...,  , 's...	Car Polygon, textured, smooth shading, normals
815	815,  ,  , 	Line
81B	81B,  , count,  , 's...	Polygon, flat shading
81C	81C,  , count,  , 's...,  , 's...	Polygon, smooth shading
81D	81D,  , count,  , 's...,  , 's...	Polygon, normals
81E	81E,  , count,  , 's...,  , 's...,  , 's...	Polygon, smooth shading, normals

81F	81F,0,  ,count,  's...,  's...	Polygon, textured
820	820,0,  ,count,  's...,  's...,  's...	Polygon, textured, smooth shading
821	821,0,  ,count,  's...,  's...,  's...	Polygon, textured, normals
A03	A03,  ,  ,  , 6.8, 123	SRB Selector

Display Classes

These are used (I think exclusively) by the texture selection nodes, T-5xx. They define classes of texture, and thus polygon, so the graphics detail level can be reduced. Some are car only.

Number	Type of thing in class	Menu option that affects the class
0	Track	Track Textures
1	Grass, curb, edge, bank	Terrain Textures
2	Fence, armco	Wall Textures
4	 (external)Engine	Car Textures
8	Trees?	??
10	Startline, starting grid boxes	?Track Textures?
13	 Face, rim, shock, etraymap	?Car Textures?
17	 Steering	?Cockpit Texture?
18	???insidelf	??
19	 Knees, driver, arms, neck	Driver Textures
20	Racing groove	Racing Groove

Detailed Descriptions

General Nodes

T-3 .SRB File

3, vertex number, colour, string offset, float, int

Colour may or may not be a colour. I've only seen it as 0xffffffff, so it might be a pointer (never used?)

Vertex number suggest the centre of the SRB, i.e. as if it were an object in 3-space. String offset gives the filename of the .SRB file, when looked up within the String section (string header not included).

The float looks like it defines a scale, and the int seems to be a bit pattern (e.g. 65536, 0...)

T-4 Multi-child node

4, number of offsets, offset...

This type acts as a simple multi-root node.

*T-5_{1,4,5,8,c,d,11,1c,20,24,2c} Texture Selection

5, offset, sub-type, other data....

T-5₁

5, offset, 1, string offset

String offset points into the String section, and is used to choose a MIP file for the textures.

T-5₄

5, offset, 4, number

Only ever seen as the root node. The number has unknown meaning.

T-5₅

5, offset, 5, String offset, number

String offset points into the String section, and is used to choose a MIP file for the textures. The last number has an unknown meaning (probably something to do with the graphics options).

T-5₈

5, offset, 8, int1, int2, float

I'd have thought one of the int's specified a string offset but have been unable to determine which. The float is always(?) 0.3 or 0.6.

T-5_{c(dec 12)}

5, offset, 12, int1, int2, int3, float

See sub-type 8.

T-5_{d(dec 13)}

5, offset, 13, string offset, int1, int2, float

See sub-type 8 (but this time I know the string offset is where it is :-)

T-5_{11(dec 17)}

5, .3do offset, 17, number of .mip string offsets, .mip string offsets...

This looks like it chooses a single .3do file, and several .mip files. I've seen it used in the lotus wheel .3do's (of which there are many).

Sometimes, the .mip offsets are the same. I believe this means that two different file extensions are used, e.g. file.mip & file.3do. This would be a bit bizarre, because usually the .3do that contains this T-5₁₁ has the same filename, so it would be calling itself. Or it could be that these multiple calls reference the HAND sections of the referenced 3do??

T-5_{1c(dec 28)}

Haven't done this one yet...

T-5_{20(dec 32)}

5, offset, 32, int

The int is a number used to choose a predefined texture, possibly from a list of mips chosen by another node. This node has only been found in car wheels so far, where the int is used for: 0 or 1 = tire wall texture, 2=tire tread texture.

T-5_{24(dec 36)}

5, offset, 36, int (0), int (4)

Looks suspiciously like sub-type 4, so the first int is probably a string offset, but I can never tell when it's zero :-)

T-5_{2c(dec 44)}

5, offset, 44, int (0), int (4), int (0xA), float (0.3 or 0.6)

Mmm...

A side note:

Plane calculations:

To calculate which side of a plane a point (i.e. viewer) is on, find d ($d=Ax+By+Cz+D$), where $ABCD$ are the coefficients of the plane equation (i.e. X,Y,Z,S) and xyz are the coordinates of the point.

If $d<0$, point is 'below' the plane

If $d=0$, point is 'on' the plane

If $d>0$, point is 'above' the plane

Where 'above' is taken from the plane in the direction of X,Y,Z , and below is in the opposite direction.

T-6 Single child with plane

6, plane number, offset

The plane number references the PLAN section. The offset contains the polygons visible when the viewer is 'above' the plane.

T-7 Double child with plane

7, plane number, offset, offset

A dual child node, with a plane. Each offset is followed only if the contents can be seen.

(almost) **T-8 Bi-directional version of T-9/T-A**

8, plane number, offset1, offset2, offset3, offset4

This behaves like T-9 and T-A, in that the first and last offsets are the sub-trees 'above' and 'beneath' the plane. The second and third offsets are the sub-trees 'on' the plane, one viewed from above, the other when viewed from beneath.

This node is used in places like a fence bordering the track, where the fence can be viewed from both sides.

T-9 Three child with plane

9, plane number, offset1, offset2, offset3

The plane number references into the PLAN section. One or more of the offsets is often NULL (i.e. 0xffffffff).

The first offset is for things that can be seen 'below', the second for 'above', and the third is always followed, as far as I can tell (above/below, that is).

T-A Three child with plane

Same as T-9

No idea why T-A and T-9 are the same, so there must be some difference. Possibly to do with the plane number, but more likely to do with the context they appear, i.e. what Types do they point at and are pointed at from? Or maybe, T-A and T-9 are the same, but the sub-trees are the opposite way around - i.e. above=below, below=above?.

T-B Double child with plane

B, plane number, offset1, offset2

A simple 2 child node with a referenced Plane. So why's it different to T-7?

Possibly that T-7 contains 'above the plane' and T-B has 'beneath the plane', or the other way around.

T-D Positioner

`D, dx, dy, dz, rotx?, roty?, rotz?, scale, offset`
7 floats, suggesting 3 for xyz, 3 for rotations & 1 for a scale (usually 1.0 - i.e. not scaled).
The three rotations are usually in the range -6.28 to 6.28, i.e. -1radian to +1radian.

This type seems to be a general transformation node - that is, all children of this node are changed by this node.

T-E 3DO file

E, String offset, number

number has only been seen as 0

String offset gives the name of another 3DO file. T-E's seem to follow from T-13's and I believe are used together to define the position of an external object.

T-F Track visibility box

>> Doh! There's only four plane/offset pairs, then an offset, then -1 (Thanks Nigel!)

```
F, plane1,offset1, plane2,offset2, plane3,offset3, plane4,offset4,  
offset5, 0xFFFFFFFF (-1) 0,0,0,0, no. of following sets, sets...
```

where a set is: float, float, float, offset.

Where planes & offsets can be 0xffffffff and shouldn't be followed if they are.

I.e. 4 zeroes indicates there is no more data, 4 zeroes and a 3 indicate there is 3*16 bytes more. The three entries listed as float could be x,y,z coordinates but I'm just guessing.

An example of a T-F would be F, (p,o), (p,o), (p,o), (p,o), o,-1, 0,0,0,0,4, (x,y,z,off), (x,y,z,off), (x,y,z,off), (x,y,z,off).

The offsets contained with the *sets* point to 3DO chooser nodes - T-13's. I.e. a T-F might point to 4 other T-F's and 2 T-13's.

These nodes should not be treated as part of a single large tree. They create loops if treated this way. For more information see the [Track](#) part of this page.

T-10 Chooser?

This appears at the start of the PRIM section, the end, and in the collision part of a 3DO among other numerous places.

```
10, number of numbers, numbers...
```

T-10's appear to choose a set of numbers, but as they are used in different places, the meaning must need to be taken in context. I.e. T-10's are used in collision trees, for .trk numbers, for sounds, etc.

T-11 Distance Switch

```
11, number1, vertex, number3, no. of pairs, pairs(float, offset)...
```

Number1 would appear to be either 0 or 4. If 0, the floats given in each pair are in decreasing size. If 4, all the floats are zero. Number3 is either 2 or 5 and usually 2 goes with 0 and 5 with 4, but there are some exceptions.

The vertex is the reference point used when calculating how far away the viewer is.

If the node was: 11, 0, 123, 2, 3 pairs, (100, tree1), (50, tree2), (0, tree3)
Then from infinity to 100 metres away then tree1 will be followed; from 100 to 50 metres tree2 will be followed and 50 to 0 metres tree3.

T-13: Place a 3DO object

13, dx,dy,dz, rotx?,roty?,rotz?, scale, offset

7 floats and an offset. The offset points at a T-E, which chooses a 3DO file.

T-14 Unknown

Found in lotus/lotus.3do & presumably other cars.

14, number, number.

So this Type looks like T-10 with unknown numbers

T-16 Positioner

16, float, float, float, float, float, float, float, offset

Found in lotus/lotus.3do. Almost identical to T-D & T-13, with 7 floats and an offset

T-17 Room?

17, no. of vertices, vertices...

Found in lotus/lotus.3do & other cars. Seems to define an area via vertices.

T-19 Room?

19, v0, v1, v2, v3, v4, v5, v6, v7, offset

The v's are vertex numbers. When drawn, these vertices create a 6 sided enclosed region:

[v0-v1-v2-v3], [v4-v5-v6-v7],

[v0-v1-v5-v4], [v2-v3-v7-v6],

[v0-v3-v7-v4], [v1-v2-v6-v5].

T-A03 SRB file?

A03, vertex number, colour, string offset, float, int3

Looks like T-3 but for the cars. Untested.

Lines/Polygons

T-1f Polygon - Textured

1f, 0, number (hex 05000000 or 0a000000), no. of vertices, texture x,y pairs..., vertex numbers...

Looks remarkably like T-81f, just with an extra number at the start. Only found in car 3do's.

T-20 Polygon - Textured, Gouraud

20, 0, 0, number (hex 01000000), no. of vertices, texture x,y, pairs..., vertex numbers..., vertex colours...

Looks remarkably like T-820, just with a few more numbers at the start. Only found in car 3do's.

T-21 Polygon - Textured, Gouraud, Specular

21, 0, number (hex 02000000), no. of vertices, texture x,y, pairs..., vertex numbers..., vertex normals...

Looks remarkably like T-821, just with an extra number at the start. Only found in car 3do's. The last set of numbers are normals for lighting effects.

T-815 Line

815, colour, vertex1, vertex2

Probably the simplest primitive, 815's define lines. These are used for things like telegraph wires and don't appear very often.

T-81B Polygon - Flat

81B, colour, no. of vertices, vertex numbers...

Defines a polygon. No texture mapping, flat shading.

There are no. of vertices vertex numbers at the end of this Type, e.g. 81B, FF204080, 4, 12,13,14,15.

T-81C Polygon - Gouraud, No Texture

81C, colour, no. of vertices, vertex numbers..., vertex colours...

Defines a polygon. No texture mapping, possibly Gouraud shading (from the vertex colours).

T-81D Polygon - Specular

81D, colour, no. of vertices, vertex numbers..., vertex normals...

Defines a polygon, No texture mapping, possibly fancy shading (using normals). If no. of vertices is 6, there are 6 vertex numbers and 6 vertex normal numbers. The numbers reference the XYZS & NORM sections.

T-81E Polygon - Gouraud & Specular

81E, colour, no. of vertices, vertex numbers..., vertex colours..., vertex normals...

Defines a polygon, No texture mapping, Gouraud shading. The normals are often all the same, but this is acceptable since they specify direction from a vertex.

T-81F Polygon - Textured

81F, 0, colour, no. of vertices, x,y pairings for each vertex..., vertex numbers...

Defines a polygon. Texture mapping using the xy pairings (floats), flat shading if not textured.

T-820 Polygon - Textured, Gouraud

820, 0, colour, no. of vertices, x,y pairings..., vertex numbers..., vertex colours...

Defines a polygon. Texture mapping, Gouraud shading if not textured.

T-821 Polygon - Textured, Specular

821, 0, colour, no. of vertices, x,y pairings..., vertex numbers..., vertex normals...

This looks like a polygon with texture mapping, flat shading, specular shading.

Handles

.3DO File Format - Handles by P.A.Flack 2000

last revised 2-June-2000

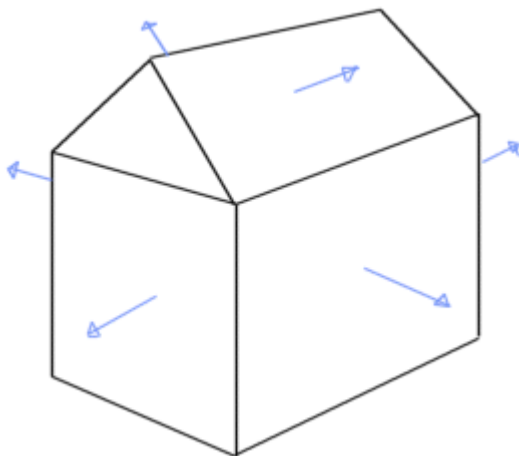
HAND - Handles

This section is used to split the Primitives (PRIM) section into sections. It consists of a set of string offset/offset pairs into the STRN & PRIM section, from where the normal reading can take place. The 3do file must also be read normally, from offset 0 -> offset x to get the main visible part of the 3do. The Handles seem to define things like collision trees (as in watglen\hay01.3do) or sub-objects (lotus.3do), etc. Work remains on this section & STRN section, especially in the car 3do's, as things like ">" and numbers are referenced, suggesting a kind of script language.

2-June-2000: Time for some more information!

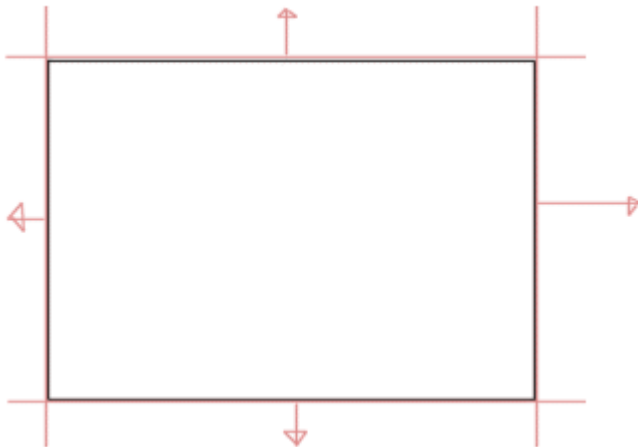
As already mentioned, Handles allow for multiple trees within the PRIM section. Here I will explain how this can be used to make an object solid, by adding a collision volume.

To make an object solid, we must define its sides - e.g. for a house we'd have 4 walls, and 2 'sides' for the roof. This will enclose the entire house behind the 6 'sides', which are actually planes.



With those 6 planes (arrows represent the direction of the plane), the house is enclosed. If we were to test a point in space against these planes, we notice that when the point is behind *every* plane, it is actually inside the house - i.e. in collision with the house. If the point is in front of even just one of the planes, a collision has not occurred.

Here's a plan view of the 4 walls of the house, where the red lines/arrows are the 4 planes:



So how does this fit into the 3DO ?

Like this (assuming you're familiar with the nodes):

Node 4, 1 child

Node 4, 6 children (one for each of the planes)

Node 6 (referencing the first plane)

Node 10, with 1 number

Node 6 (referencing the second plane)

Node 10, with 1 number

Node 6 (referencing the third plane)

Node 10, with 1 number

Node 6 (referencing the fourth plane)

Node 10, with 1 number

Node 6 (referencing the fifth plane)

Node 10, with 1 number

Node 6 (referencing the sixth plane)

Node 10, with 1 number

See how there is one type 6 -> type 10 node, per plane. The number in the type 10 node defines how the cars collide with the object - e.g. soft, solid, bouncy, sticky, etc. (not sure of the exact numbers...)

Some notes:

* You can only define concave collision volumes - e.g. a cube is okay, so is a sphere, but a cube with a hole in the middle is not okay, because the surfaces double back in on themselves, creating a confused object.

* But ! You *can* have multiple collision volumes in each object. See the tree above - the first node has only one child - e.g. one collision volume. By having multiple children (each the same layout as the example), you can create collision volumes such as the cube with a hole. E.g. a bridge generally has two uprights, and a cross piece. By using 3 collision volumes, this will work nicely.

* It is not necessary to completely enclose an object - if a cube is standing on the ground, why bother enclosing the bottom ? The cars cannot go underground ! Similarly, something like a flag pole can do without a top, because the chances of a driver landing on top of the pole are fairly remote :-)

.3DO File Format - General 3DOs by P.A.Flack 1999

last revised 7/June/1999

Most 3DOs, that is, 3DOs that aren't tracks or cars, have the following structure, where some of the sections are optional (and can probably be in any order):

Header; Normals; Vertices; Planes; Strings; Primitive Tree(s); Handles

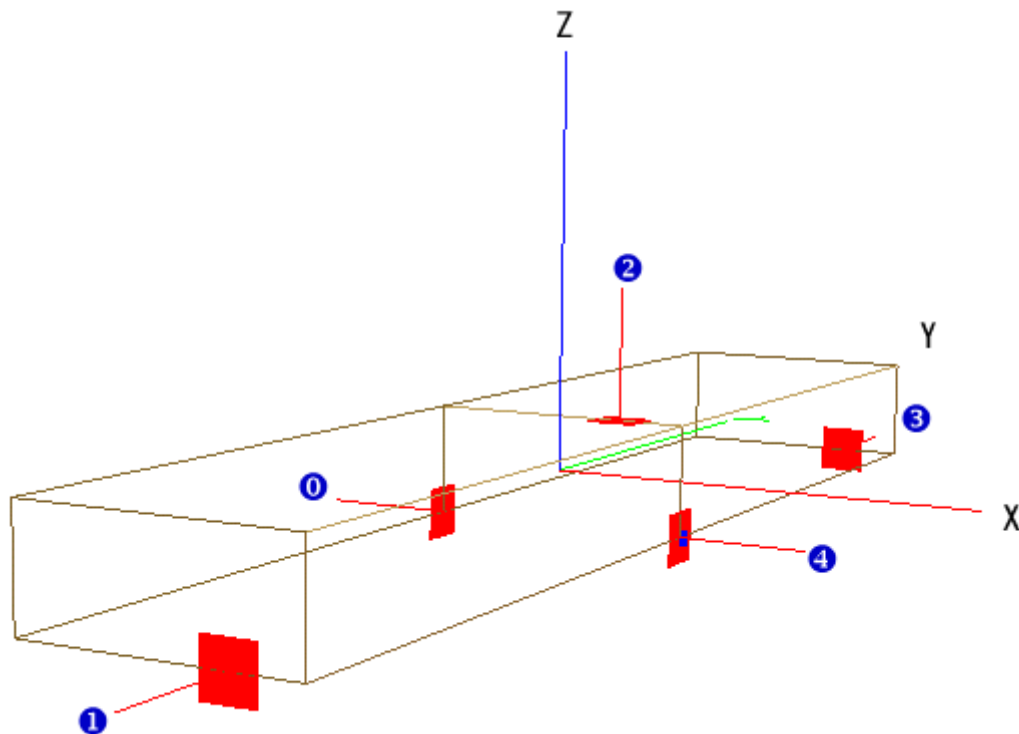
This page will hopefully explain the way in which these work together to create the objects used in GPL. Hopefully. If you are new to 3DOs, I recommend you take your time and read all of the pages here several times over before doing anything. Unless you think it's all too easy ;-)

As you'll find out (if you haven't already), 3DOs are complicated. Especially the cars and tracks. I will be using an example 3DO familiar to many, the hay bales at Watkins Glen. This is a good choice because it has relatively few polygons, and has a collision detection volume built in. Once you've mastered this, the other features that 3DOs have will be childsplay (sort of).



The 3DO is a cuboid shape, with 8 polygons (no base, 2 per long side/top). Here is the same object as shown by GPLEditor (by Paul Hoad). Note the positions of the axes, Z is up (completely wrong in my opinion, as Z should be depth and Y should be up, but that's

Papyrus' choice and enough of that debate).



I'll assume an understanding of 3D vertices and the fact that polygons can be constructed from those vertices. In the hay bale (bale?, bail?), only rectangles are used.

Texture Mapping

To make everything look lovely, nearly every polygon in GPL is texture mapped. Those that aren't are coloured with single colours (every polygon is, but usually the texture mapping is chosen above flat shading). For each polygon, the appropriate texture needs to be chosen, and texture coordinates are needed to specify where the texture is placed on the polygon.



Lets say that this is the texture to be applied to a polygon. Texture coordinates usually range from 0.0 to 1.0, where 0 is top/left and 1 is bottom/right. (Either or both of them might be the other way around, I can't remember :-). If a texture is to be repeated, values greater than 1.0 are used. If only a small piece of the texture is to be used, values smaller than 1.0 are used. Negative values are never used. If the values used are the wrong way round, the texture will be mapped back-to-front.

Hidden Surface Removal

Drawing the polygons aside, this is the most time consuming task. It is necessary to remove the hidden surfaces because if they weren't, the number of polygons to draw each frame would be too large in comparison to the number seen, and performance would be badly affected.

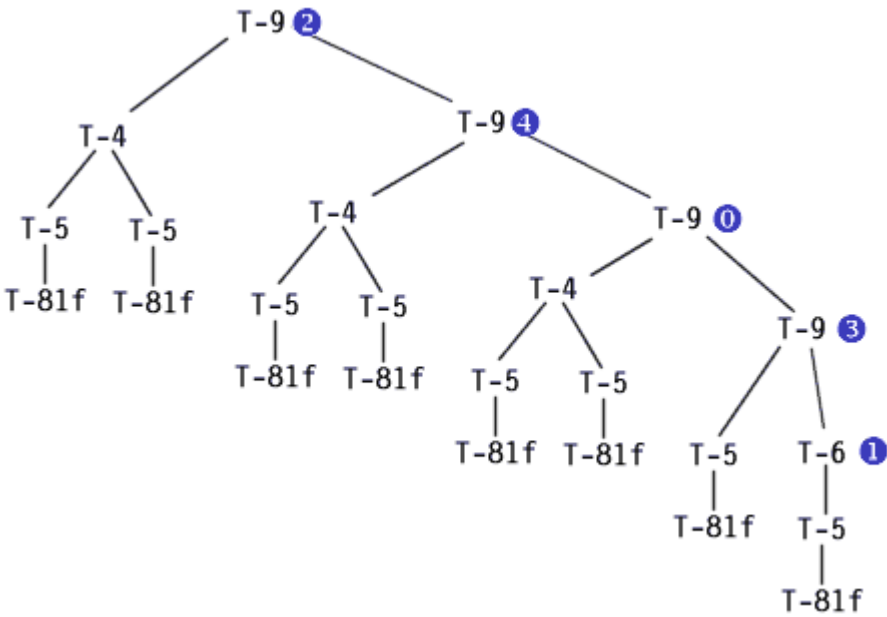
Take a wall that has two sides (forget the thickness for now). When you are standing on one side, unless you have some really cool vision, you can't see the other side (also forget mirrors

and such). If you move around to the other side, you can't see the side you were on. Thus, to reduce the number of polygons drawn by half, all that is needed is a way to determine which side of the wall you are on. The same goes for objects which are on opposite sides of a surface. If you have an apple behind a wall, it is fairly safe to say that it won't be visible from in front of the wall (unless the wall has lots of holes, in which case it is pointless to use visibility determination like Hidden Surface Removal - a depth buffer is much more useful - but forget this little side point if you're confused already).

To determine which side of a plane (an infinitely thin surface, extending to infinity in all other directions), a point (i.e. an observer) is on, maths is used. The equation of the plane ($Ax + By + Cz + D = 0$) is jiggled about a bit with the location of the viewer and a number results. If negative, the viewer is one side, positive the other, zero and the viewer is actually on the plane (and thus in a bizarre infinity reality Star Trek situation and can't see anything (or can they?)).

Back to the hay bale, and you'll see 5 numbers from 0 to 4. These reference the red squares/lines, which are the planes for the hay bale. Most planes you'll encounter will be in the same plane as a polygon, like those in the example. It is not necessary for this to be the case, but it usually helps. The numbers are the same as used by GPL.

Now we need to see the 3DO as a tree:



Yes, it looks scary. Basically, everything beginning with a T- is a node in the tree. The T-81f leaves are the polygons. The T-5 nodes choose the appropriate texture. T-4's act as a branch and can have any number of children. The important nodes for this tutorial are the T-9's and the T-6. These are the nodes which reference the planes (numbers shown).

When GPL comes to render the hay bale (or any other object), it starts at the top of the tree (T-9,2) and sees which side of plane number 2 the viewer is on. Looking at the diagram, it can be seen that if the viewer is above the plane (in the direction of the line), they can see the contents of the left hand branch (which is actually the middle branch, because no left branches are defined for this particular object). Presumably, if they are below the plane they would be shown the contents of the left hand branch. The rendering then continues by following the

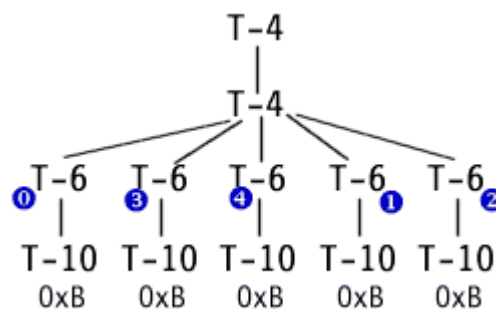
right hand branch, where similar checks are made.

The T-6 node is a bit different in that it does not contain anything other than a single branch which is followed if the viewer is in front of the plane, otherwise is isn't.

Collision Volumes

The planes are not used exclusively for visibility determination. They are used for detecting whether collisions have occurred.

Remember that planes are infinite, and divide space into two. If you take several planes, it is possible to define a region of space which is behind all of the planes, i.e. enclosed. GPL uses this (not enclosed top and bottom, though), to define volumes where collisions occur. This is achieved by using a tree of a particular structure:



This tree is contained within the hay bale's 3DO as a separate tree. A Handle is used to point to the tree, under the name 'collision'. (see the [Handles](#) section for more information). The number of children in the first T-4 defines how many volumes make up the collision detection, for there can be more than one. For each volume, a T-4 has n children, one per plane, that consist of a T-6 (with the appropriate plane id), and a T-10, which lists numbers. For collision volumes, there is only ever 1 number, for this example it is 0xB, or 11 decimal, which presumably means hay, or more accurately, a solid collision. I haven't yet researched the different types of collision available. Needless to say there are types with different rebound characteristics, and types like hedges, that 'catch' the car.

a sort of tutorial, which doesn't quite fit in with the rest of these pages...

 [Tracks](#)

.3DO File Format - Track 3DO's by P.A.Flack 1999

last revised 7/June/1999



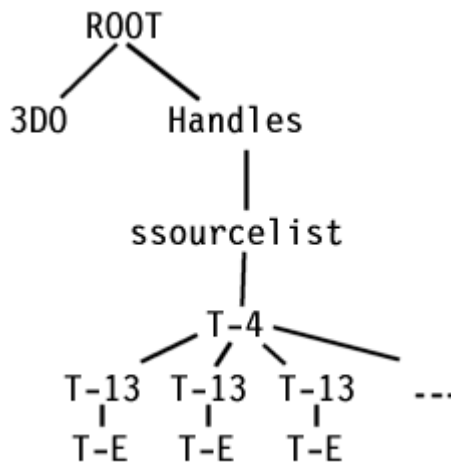
07/June/99: Figured out the meaning of the first big T-10, the little T-10's and most of the second big T-10; a few rewrites.

24/May/99: Started as a new file in its own right.

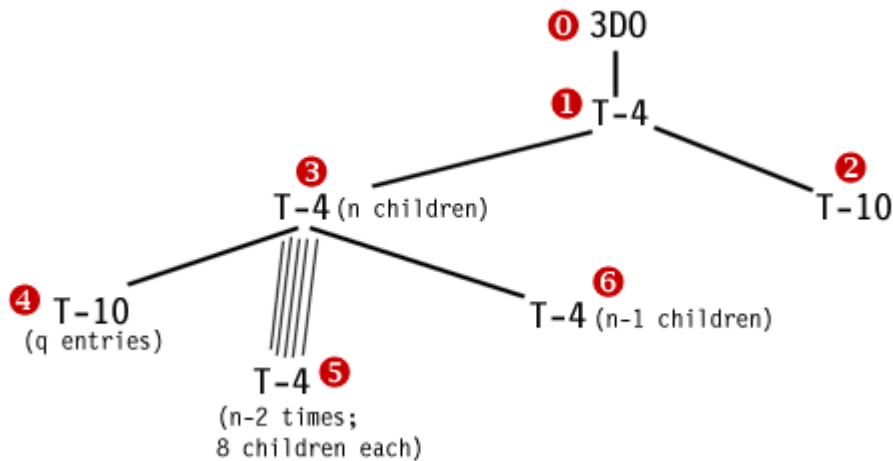
The structure of the track 3DO's is very complex, uses the 3DO nodes in a very strict fashion and is closely linked to the .trk file. There is still a some information be determined about the tracks...

A note about numbers. TRK numbers are large integers (i.e. 20100000). These numbers are actually 1/500ths of an inch, so to turn them into metres (as used by the 3DO's), divide by 19685.03937, or more accurately, 31680/1.609344. The track 3do's use TRK numbers in several places, probably to confuse anyone looking at the files...

This diagram shows the layout of kyalami.3do:



The part labelled 3DO is expanded in the next diagram. Handles is the HAND section of the track, and I have only seen it containing one entry, that referencing the ssourcelist in the STRN table. This would suggest sounds. The T-4 (and sub-structure) is the only element to this handle. The T-13 and T-E combinations choose sub-3DO files. In the case of kyalami, at least, these 3DO's are the grandstand 3DO's. Strangely, the track.3do's do not contain direct references to the sound files. Instead, the sub-3DO's (gstand1.3do, for example), also have HAND sections with ssourcelist entries. In these, there is a T-10 which chooses numbers, presumably the id for the relevant sound.



This is where the tracks get interesting, hehe...

1 is a T-4 node with two children.

The second of these, **2**, is a list of numbers. The number of these is always a multiple of 4 (for all the standard GPL tracks, anyway), and is equal to the track length in metres divided by 39.0m (roughly). Kyalami for instance is 3970.49m in length and has 404 entries in the list. $3970.49 / 101 (101 \text{ groups of } 4 = 404) = 39.3\text{m}$. Thus there is one group of 4 numbers every 39.0m around the track (39.3 in Kyalami's case, but that is probably due to 3970.49 not being a multiple of 39).

These groups of 4 are as follows:

The first number appears to be a distance (trk) that things can be seen when looking forwards. The third number appears to be the same, but when looking backwards.

The second and fourth are nearly always zero, except at locations like hairpins, where two pieces of track can be seen at once. I believe that the second must be the distance forwards down the other piece of track (so that would be like looking backwards down that piece of track), and therefore the fourth number must be backwards (forwards) down the other piece of track. I'm not convinced about these numbers yet... It would appear that GPL accepts the 4-tuple as 10000000,0,10000000,0 without much complaints (haven't tried this as hairpins...)

3 is a T-4 with n children. Of these, the first is **4**, the next n-2 are **5**, and the last is **6**.

4 is a T-10 node with q entries (the letters n and q are entirely not significant). The value of q is obtained by dividing the track length by 13.3. This is because there one number for every 13.3 metres (roughly) around the track. The values of the entries are from 0 to n-3, i.e. there are entries for every entry in the **5** structure. The example track, Kyalami, has 299 entries in this part of the tree, and they go like this: 0, 1, 2, 2, 2, 2, 3, ..., 88, 88, 89, 89, 89, 89, 89. Kyalami has n as 92. Thus for Kyalami, if the car is in the first 13.3 metres of the track, the first 3do section will be used, all the way up to $16 * 13.3\text{m}$, then the second 3do section, then the third for $4 * 13.3\text{m}$, etc. Sometimes the 3do sections will not be listed in this list, because they are too short. This leads me to believe that GPL uses the 3do sections surrounding the one the car is in, as well as the one the car is in.

6 is another T-4, with n-1 children. These children are all T-F nodes. As will be explained later, these are decidedly nasty...

5 is a set of n-2 T-4 nodes. Each of these nodes have 8 children:

Child no.	Child Node types	What is it?
1		
2		
3	T-B, T-8 or NULL	highest level of detail
4		
5		
6	T-B, T-8 or NULL	middle level of detail
7	T-B or T-8	far away level of detail
8	T-10	a list of 4 TRK numbers

These nodes define 3 levels of detail of track graphics. When viewed from far-away, this section of track (i.e. Kyalami has 90 visible track sections) is represented by one region, as

defined in the sub-tree 7. When viewed from closer to, the section is broken into two regions, 5 and 6. Closer still and the section consists of 4 regions, 1-4.

It is not necessary to define region 6 - it can be a null entry (5 is always required). If 6 isn't defined, regions 3 and 4 will never be defined. If 6 is defined, region 4 may be non-existent. Similarly, region 2 may be non-existent. If regions are not defined, the regions that are defined must cover the regions that are not.

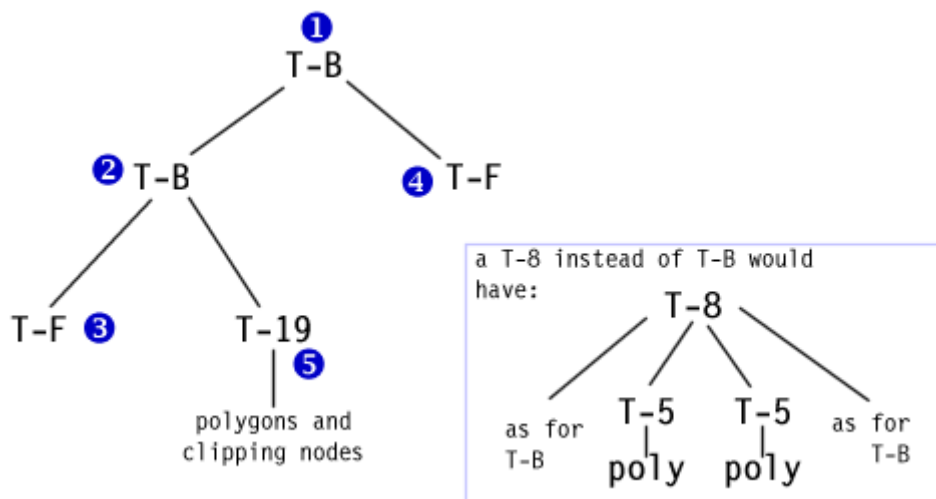
To get it out of the way, the last entry, 8, is a T-10 node with 4 entries. These entries are TRK lengths, i.e. divide by 19685.xxx to turn them into metres. They give the starting position around the TRK line of each region. For straights, all 4 numbers will (nearly always) be the same. Corners usually have 4 different numbers, the first for the first region, second for second, etc.

Regions, for want of a better description:

These contain all the graphical details that we're already familiar with - polygons, as well as some T-F nodes, T-19's and some other bits and pieces.

Think of a region as being a segment of track, from the far left wall to the far right wall, of say distance 12 metres (just an example distance, each region is usually a different length). The 7th region will thus cover all of this area. The 5th and 6th will cover 6 metres each, and the 1st-4th will cover 3 metres each.

The node that starts off the section is one of two types (ignoring the null entries now), T-B or T-8. The difference between the two is that T-B is used where there is not a wall, fence, etc., and T-8 is used when there is. T-8's - not used elsewhere very often - are used to split the region into four - the two bits of land either side of the wall, and the two sides of the wall. There are two T-8's/B's and lots of other bits, best explained with another diagram:



(Nodes ① and ② can be replaced with the bit in the box, for when walls are present).

I do not know which of ① and ② is left and which is right. I believe the first is the left, but it probably all depends on how the planes and sub-structure is all oriented. The same goes for the two polygons in the T-8's - one is left and one is right...

Anyway, on with the structure.

③ & ④ : Outside the walls (or lack of walls - think of them always being there, just invisible. They are not the edge of existence, just the edge of the track) is the region in which the trackside objects exist - all referenced via the two T-F structures.

⑤ : In between the 'walls' is the visible part of the track - all defined under the T-19 node, which is a volume defining node. I believe this is to define the region in which cars are supposed to be. It is 3D and covers the entire region between the two walls and start and end of the region. The sub bits of ⑤ are all the sort of 3DO structure that is used everywhere, apart from the fact that the T-F nodes rear their ugly heads again. This allows for sub-3DO's to be placed within the walls. I believe there is one sub-region in this part of the structure per bit of the T-F structure.

And now for something completely scary...

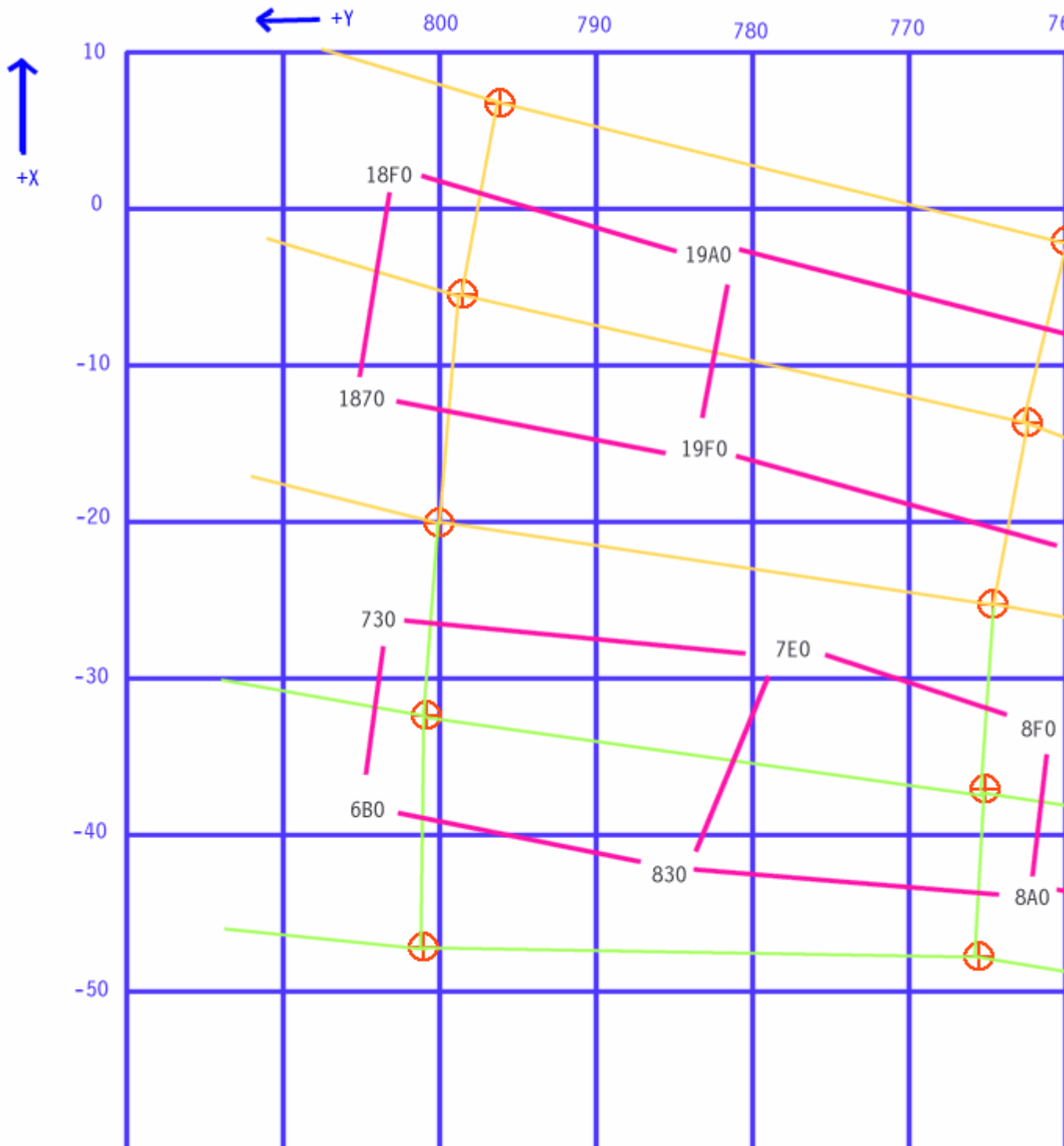
T-F's!

Unfortunately I cannot get away without explaining these, as they are so well integrated with the rest of the structure that to leave them out would be stupid. Grrr... T-F's tell GPL where external 3DO objects are located. They work as a grid covering the entire visible area, and even extending off to infinity track-left and track-right. Thus 3DO's can be placed anywhere by adding them to the appropriate T-F node (or nodes - a grandstand might be in more than one T-F...)

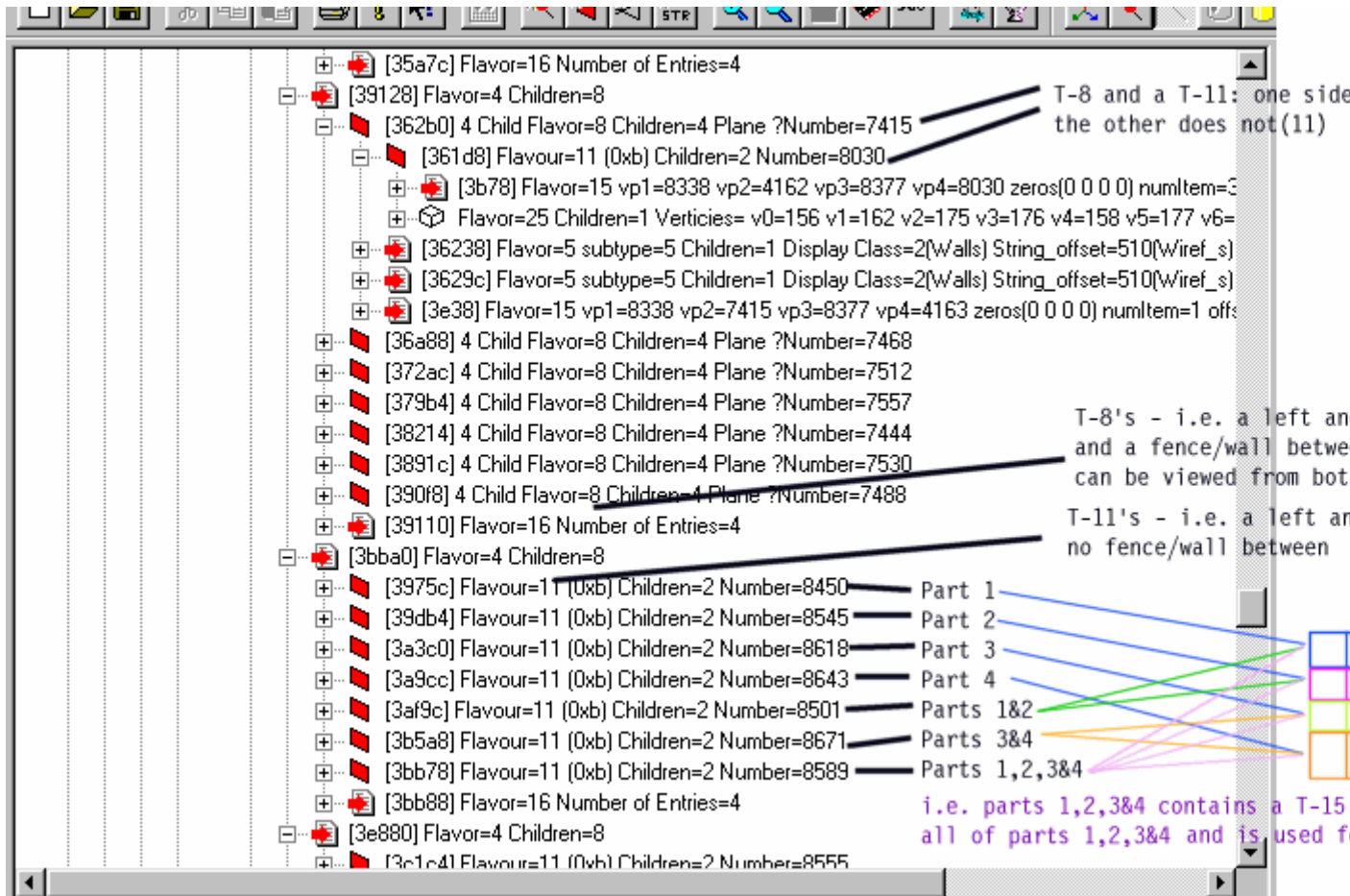
OK then, This huge diagram shows the T-F areas from Watkins Glen (the first few, covering the starting line onwards). As you can see, I drew this back in February, but the comments down the side are still relevant.

What I haven't said is that each area is defined by 4 planes. The space within each of these is the area defined. (?)

The actual polygons and so on fit into the confines of the areas, but there may be more than one polygon per area - e.g. a bit of track, kerb and a bit of semi-transparent racing line.



Lastly, for now, this diagram shows pretty much all of what I've been explaining, in GPLEditor by Paul Hoad. Please note that Paul uses decimal notation (not that that's bad in any way) for the node types (called flavours below), so T-8 = flavour 8, T-B = flavour 11, etc.



Grand Prix Legends .MIP File Format by P.A.Flack 1998

contact: phil@flack99.freemove.co.uk

last revised 21/Dec/98

MIP files, or Multiple Image Pictures are used for textures requiring different resolution versions of the same picture, so that at a distance there is less data to display, and up close the image looks good. All MIP's start with the same header:

Bytes	Values	Notes
0-3	"PIM "	Section Type (last char is a space - 0x20)
4-7	0x00000000	0
8-11	data size	size of following data

After the header section comes the real header section:

Bytes	Values	Notes
0-3	"DHPM"	Section Type
4-7	0x00000000	0
8-11	data size	size of following data (generally 0x12 - doesn't include padding bytes 30&31)
12	0,1, 2, ?other?	Specifies type of MIP file (see next section)
13-16	power width	if this is e.g. 8, maximum images width is $2^8=256$ pixels

17-20	power height	same but for height
21-24	average colour?	seems to be the average colour, for when the image is drawn tiny (1x1 or 2x2 pixels)
25	unknown- 0,1,2,3	can't find a reason for this - only seen as 0, 1, 2 or 3 (0/3 mostly)
26-29	no. of images	number of images in the file
30-31	padding	two 0x20 padding bytes to get size up to a multiple of 4

The MIP type (byte 12 of the MPHD section) is either 0, 1 or 2, but maybe some other values:

0: Image is completely opaque, i.e. all CMAP (later) entries have the alpha channel set to 0xff

1: Image has transparent parts, i.e. CMAP entries range from 0x00 to 0xff.

2: Each image has it's own CMAP, with no general CMAP, i.e. each image has a different palette.

other: Each image is 16bit (?)

- the only check needed at this stage is if the type is 0 or 1. If so, there follows a CMAP section

Next follows a series of Bitmaps, each with their own header, data, and sometimes CMAP sections - but this can be determined from the BMAP header.

Grand Prix Legends .SRB File Format by P.A.Flack 1998

contact: phil@flack99.freemove.co.uk

last revised 21/Dec/98

Bytes	Values	Notes
0-3	"~BRS"	Section Type - SRB(space)
4-7	0x00000000	0
8-11	data size	Size of following data

SRB's consist of the above header, and two other headers, defined below. The second of these seems to be unimportant, as the contents are all zeroes...

Bytes	Values	Notes
0-3	"DHRS"	Section Type - SRHD
4-7	0x00000000	0
8-11	data size	Size of following data (24?)
12-35	data...	data... seen as 6 floats.

Bytes	Values	Notes
0-3	"2HRS"	Section Type - SHR2
4-7	0x00000000	0
8-11	data size	Size of following data
12-	zeroes...	0's

Following the headers, there is the contents of a MIP file, seemingly without any differences to a standalone .MIP. Because of this fact, I believe the 6 floats in the SHR2 section scale the .MIP in some way. Possibly -x, x, -y, y, -z, z.

Grand Prix Legends .PBF File Format by P.A.Flack 1998

contact: phil@flack99.freemove.co.uk

These files are quite simple in structure, and are used to define pictures for use in the menus of the game (i.e. not the game itself). All this file contains is a BMAP. How uninteresting.

.TRK File Layout by N.Pattinson(edited by P.A.Flack)

contact: phil@flack99.freeseve.co.uk
or NPattinson@compuserve.com
last revised 19/June1999



15/July/99: Tutorial on creating a TRK
19/June/99: HTML'd; added pics & extra info.

Overview

The TRK file contains information about the track layout necessary to run the simulation aspects (rather than the graphical aspects) of GPL. The main facets of information are -

- track geometry
- presence of walls, kerbs etc (for collision detection)
- track surface (for grip characteristics)

The track is divided lengthwise into a number of sections. Each section is either straight or curved. A section has an associated set of walls and surfaces.

The track has a number of lines drawn along it at fixed positions across the track. A height is recorded at the intersection of each line and section boundary. This together with the curvature information from the section produces the track layout. I have called these lines traces in this document.

The data in the file is layed out as follows

1. Header
2. Trace Positions
3. Section Offsets
4. Altitude Data
5. Wall Data
6. Section Data

There are some 'magic' values used in some fields that appear to normally have a non-magic value. Such values seen so far are:

0x07070707
0x08080808
0x40000000

The following data layouts are all specified in 32bit words.

Header

Bytes	Values	Notes
0-3	"KART"	Section Type (reversed)
4-7	3000 (int)	Version (3000 for GPL)
8-11	Length (trk)	Track Length (trk number)
12-15	Traces (int)	Number of traces (<=16)
16-19	Sections (int)	Number of sections
20-23	WallSize (int)	Length of wall data
24-27	SectionSize (int)	Length of section data

TRK numbers

These are 32bit signed integers, that when divided by 19685.03937 will give a measurement in metres. Without dividing, they are a count of 2inch lengths.

Trace Positions

There are always 16 trace positions recorded in the file. Any beyond the 'Number of traces' from the header are unused. Each trace position is a trk number representing the offset from the nominal track centreline. Negative values are to the right, positive to the left.

Section Offsets

One integer for each section, being the offset of that section within the section data. Since each section's data is 52 bytes long, these numbers are just $n * 52$

Altitude Data

There is one record per trace per section. Each record is 32 bytes long, giving a total length of $32 * \text{traces} * \text{sections}$

Bytes	Values	Notes
0-3	trk	Delta1
4-7	trk	Delta2
8-11	trk	Delta3
12-15	trk	Altitude at start of section
16-19	trk	Delta4
20-23	trk	Delta5
24-27	trk	XCoordinate/Radius
28-31	trk	YCoordinate/Unused

For a given trace, the start altitude of the next section is always the start altitude of this section plus deltas 1,2 and 3 (allowing for rounding errors). I don't know why there are 3 separate values, or why deltas 4 and 5 exist.

For straight sections, words 6 and 7 are the coordinates of the trace point at the start of the section.

For curved sections, word 6 is the distance to the centre point of this curve, i.e. the radius of this altitude. Word 7 appears to be unused for curve sections.

Wall Data

NB I have named this section wall data, but it also includes information unrelated to walls.

These records are referenced by the section data - each section has its own number of wall records.

Each record is 32 bytes long.

Bytes	Values	Notes
0-3	From (trk)	Lateral position at start of section
4-7	To (trk)	Lateral position at end of section
8-11	Surface (int)	Type of surface
12-15	Unknown1	Unknown
16-19	Height (trk)	Height of wall/zero for ground
20-23	Unknown2	Unknown
24-27	0	Always zero
28-31	0	Always zero

The from and to attributes are the lateral positions on the track at the start and end of the section respectively, thus describing a line. This line may or may not be a wall. It is also used as the right margin for an area of grip which extends leftwards to the next line. Currently known surface types are:

Type	What is it?
1	Track
4	Grass
6	Gravel
10	Leftmost (unused) area of track

2048 is sometimes bitwise or'd with these values, usually when height isn't zero.

The height is the height of a wall if present.

The unknown values probably describe the wall/kerb characteristics (hedge, armco etc)

Section Data

There is one record for each section. Each record is 52 bytes long

Bytes	Values	Notes
0-3	Type (int)	Type of section (1=straight, 2=curve)
4-7	Start (trk)	Distance from start of track
8-11	Length (trk)	Length of this section
12-15	Orientation (angle)	Orientation of the start of this section (note1)
16-19	unknown1(0)	Zero?
20-23	Curve Centre X (trk)	If curve, this is the centre of the circle's x coordinate
24-27	Curve Centre Y (trk)	" y coordinate
28-31	Curvature (angle)	Curvature of this section (note 2)
32-35	unknown2	unknown
36-39	unknown3	unknown
40-43	Trace Idx (int)	First altitude
44-47	Wall Count (int)	Number of walls
48-51	Wall Idx (int)	First wall

The start of the first section is 0. The start plus the length gives the start of the next section. The total length equals the track length in the header. The orientation is the way the track is pointing at the start of the section. There are sometimes 2 adjacent straight sections with slightly different orientations.

Note 1: Orientation is an integer. When divided by 2^{31} and multiplied by 180.0, the value is in degrees.

Note 2: Curvature is an integer. When divided by 2^{31} and multiplied by 360.0, the value is in degrees.

Curvature is for curves but unknown for straights. The unit is twice the orientation unit. For a curve, the orientation of the next section is the orientation of this section plus the curvature (allowing for rounding errors).

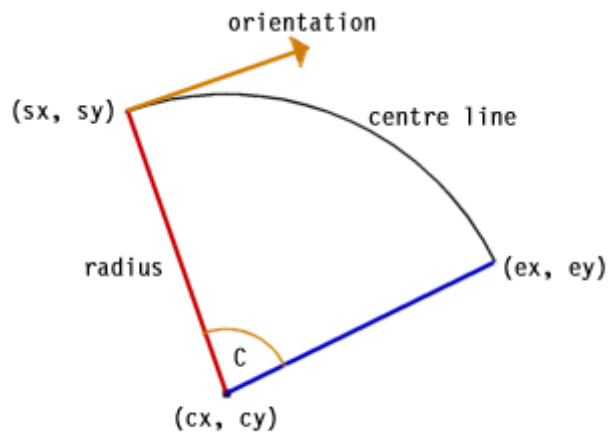
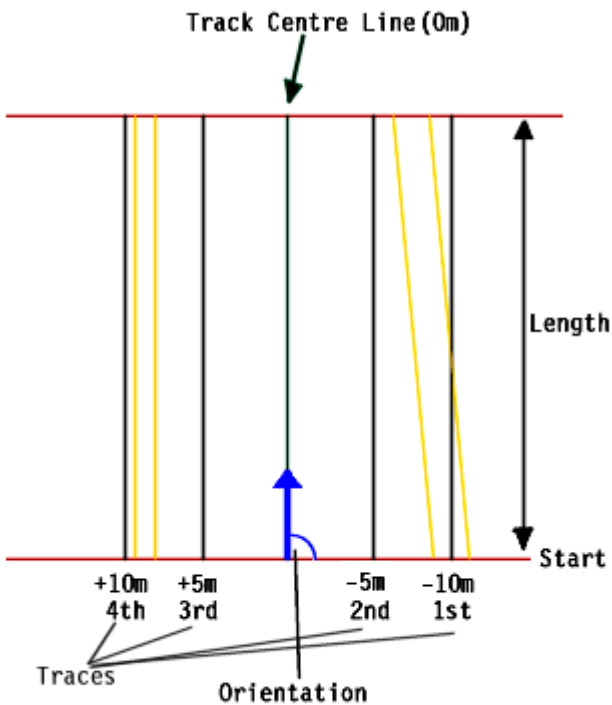
Note 3: For straights, CurveCentreX & CurveCentreY are: $\cos(\text{heading in degrees}) * (0.5 * 2^{31} / 19685.03937)$, and $\sin(\dots)$. I don't know what these values do, or even if they are important, but that's how to calculate them :-)

Similarly, for straights, Curvature is replaced by $\sin(-\text{heading}) * (0.5 * 2^{31} / 19685.03937)$, and unknown2 is $\cos(-\text{heading}) * (0.5 * 2^{31} / 19685.03937)$. Again, who knows???

There appear to be some magic values used in the two unknowns

The wall count is the number of wall records for this section. They are sequential from the wall index.

Straight Section Example



This diagram shows a straight section.

The 4 traces are black lines, and the track centre line is green. The centre line's start coordinate can be found by interpolating the coordinates of the corresponding altitudes. E.g. if this was section 4, take the four altitudes 16,17,18 and 19, and interpolate their coordinates with respect to the trace values (-10m, -5m, +5m and +10m). Next, the Orientation and Length can be used to calculate the end point.

Similar calculations can be done to draw the traces themselves, and the start and end lines (red below - +90deg and -90deg to the start and end coordinates).

Next, the walls can be drawn. In this example, I have drawn 4 walls (yellow), to represent a barrier either side of the track. The values might be

- Wall n: -11m, -8m, armco, unknown, 1.0m, unknown, 0, 0
- Wall n+1: -8m, -6m, 1 (track), unknown, 0.0m, unknown, 0, 0
- Wall n+2: +8m, +8m, armco, unknown, 1.0m, unknown, 0, 0
- Wall n+3: +9m, +9m, 10 (leftmost), unknown, 0.0m, unknown, 0, 0

Curved Section Example

Curved sections are much more complicated, but follow the same rules as straight sections. Firstly, orientation+curvature gives the heading at the end of the section. Each curved section has a cx,cy pairing to give the centre of the curve (which is an arc of a circle). Radius is given as the X coordinate of the altitudes for the curved sections. So there might be 16 altitudes, one per trace. Interpolating two of these (max. & min.) will enable the radius of the centre line to be found. From this, orientation and cx,cy, (sx,sy) can be calculated. Similarly (ex,ey) can be calculated, and so can the angles for the start and end lines (red/blue in the diagram).

Note 1: Negative and positive curvatures need to be dealt with separately (usually just a +180, or -radius, etc. difference)

Note 2: When drawing walls with different start and end laterals, the radius of the arc will not be constant, so you will need to interpolate between the start and end radii to draw the arc (I don't know of any standard graphics function that will do this though... I wrote my own :-)

Creating a TRK Pt I by P.A.Flack 1999

contact: phil@flack99.freemove.co.uk
last revised 7/August/1999

Hopefully, if you follow the instructions on this page, you will be able to create a new .trk file for the track of your choice. The method described is not the only way to get the required numbers, but in my opinion it is the most accurate, and will result in the start and end points matching up every time (good news). For the trk I used this method for, the start and end points were within 0.0001metres - not too bad...

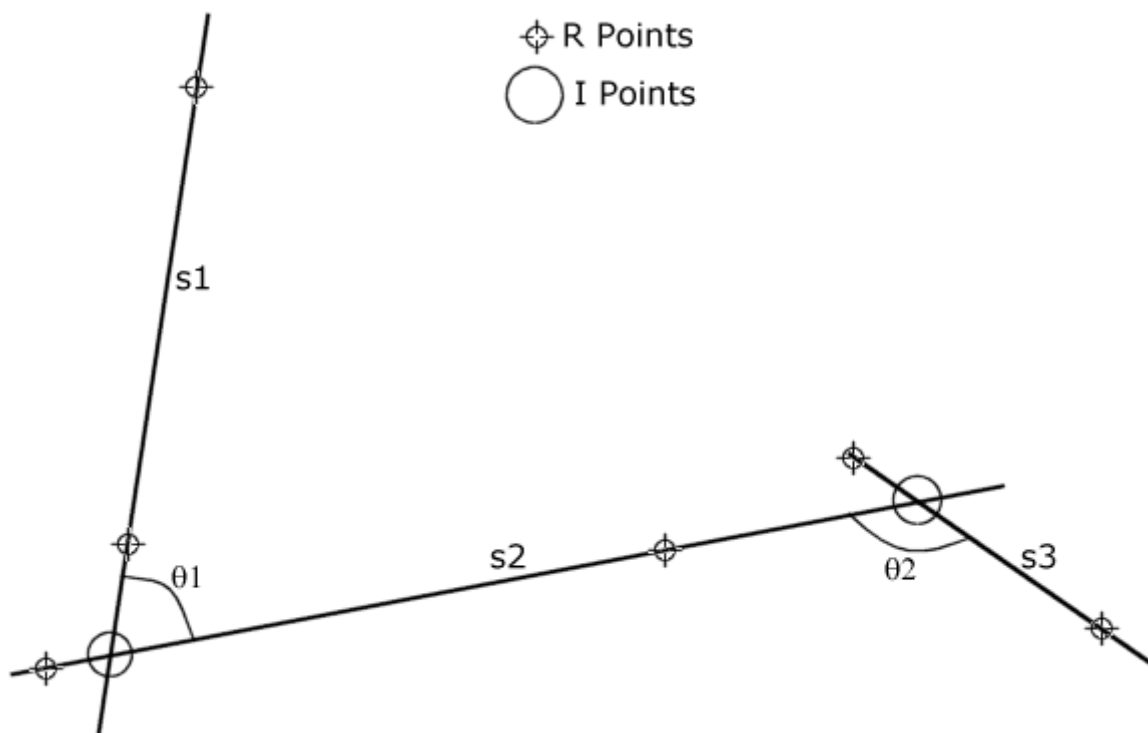
Some points to consider:

- Do not make a complex track before you've done a simple one - i.e. go for Avus instead of Clermont-Ferrand...
- This page doesn't detail how to create non-constant-radius corners. They require different centre points and I haven't worked out a mathematical method for them yet. Try to keep the corners as single sections with a straight before and after - it's a heck of a lot easier to deal with. The straights can be very short if necessary.

Stage I

Get a rough plan of the track drawn out with as many measurements as you can do on it. Typically, you'll need:

- A bunch of lines, one per straight, intersecting each other (see first diagram).
- Lengths of the straights (roughly, so you can draw the first diagram)
- Corner angles (where known - you'll be calculating them anyway)
- Corner radii (where known - you can make them up later)



This diagram shows 3 straights. The only coordinates you should know, are two (x,y) pairs for each straight (R points). These are the points that will be used to calculate everything else,

and the precision of these is not too important. It is probably easier to measure them to the closest metre, for example.

First calculations: The I points (intersections of each pair of straights). Since you know the equations of the straights (from the two points per straight), the I points can be found. Here's some code that'll do the job: You may as well use the final case for all the calculations you do by 'hand'.

```
int calcintersection(double& ix, double& iy, double ax1, double ay1, double
ax2, double ay2, double bx1, double by1, double bx2, double by2) {
    /* calculates the intersection of two lines
    /* all values are doubles to increase the accuracy
    /* parallel lines will fail (-1) (ix=0, iy=0)
    /* single points will fail (-2) (ix=0, iy=0)
    /* success returns 0
    double adx=ax2-ax1, ady=ay2-ay1;
    double bdx=bx2-bx1, bdy=by2-by1;
    if (adx==0 && ady==0) { ix=0; iy=0; return -2; } //can't do
    if (bdx==0 && bdy==0) { ix=0; iy=0; return -2; } //can't do
    if (ady!=0 && bdy!=0) {
        if (adx/ady==bdx/bdy) { ix=0; iy=0; return -1; } //can't do
    }
    if (adx!=0 && bdx!=0) {
        if (ady/adx==bdy/bdx) { ix=0; iy=0; return -1; } //can't do
    }
    if (adx==0 && ady!=0 && bdx!=0 && bdy==0) {
        //a is vertical and b is horizontal
        ix=ax1; iy=by1;
        return 1;
    }
    if (ady==0 && adx!=0 && bdy!=0 && bdx==0) {
        //b is vertical and a is horizontal
        ix=bx1; iy=ay1;
        return 1;
    }
    if (ady==0 && adx!=0) {
        //a is horizontal and b is at an angle
        iy=ay1;
        ix=((iy-by1)*(bx2-bx1)/(by2-by1))+bx1;
        return 1;
    }
    if (adx==0 && ady!=0) {
        //a is vertical and b is at an angle
        ix=ax1;
        iy=((ix-bx1)*(by2-by1)/(bx2-bx1))+by1;
        return 1;
    }
    if (bdy==0 && bdx!=0) {
        //b is horizontal and a is at an angle
        iy=by1;
        ix=((iy-ay1)*(ax2-ax1)/(ay2-ay1))+ax1;
        return 1;
    }
    if (bdx==0 && bdy!=0) {
        //b is vertical and a is at an angle
        ix=bx1;
        iy=((ix-ax1)*(ay2-ay1)/(ax2-ax1))+ay1;
        return 1;
    }
    //otherwise all other combinations must be two lines at angles, not
    parallel, nor single points. this should work since the cases where ady=0
```


or $bdy=0$ have been dealt with and so has $(adx/ady - bdx/bdy)=0$ (parallel lines)

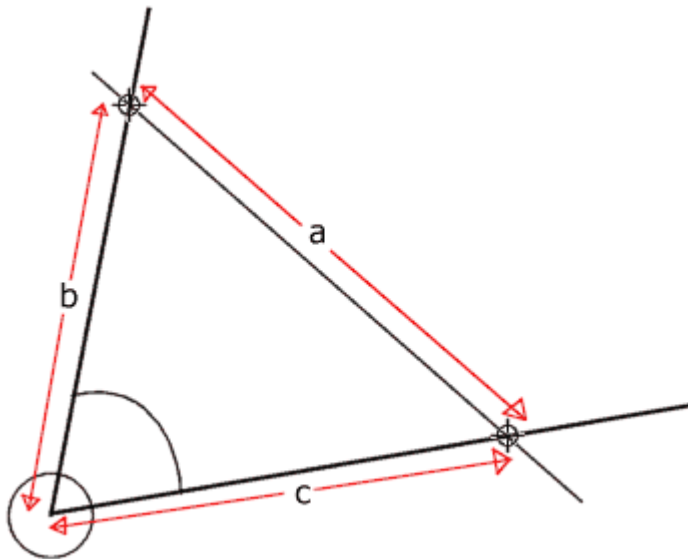
```
iy=(bx1-ax1+(ay1*adx/ady)-(by1*bdx/bdy))/((adx/ady)-(bdx/bdy));  
ix=((iy-ay1)*adx/ady)+ax1;  
return 1;  
}
```

So now you know the lengths between the I & R points ($L = \sqrt{((x2-x1)*(x2-x1)) + ((y2-y1)*(y2-y1))}$)

The angles inside the corners can also be found using the wonderful cosine rule:

$$\text{angle} = \text{acos} \left(\frac{(a^2 - b^2 - c^2)}{(-2 * b * c)} \right) \quad (\text{or: } \cos \text{ angle} = \frac{(a^2 - b^2 - c^2)}{(-2 * b * c)})$$

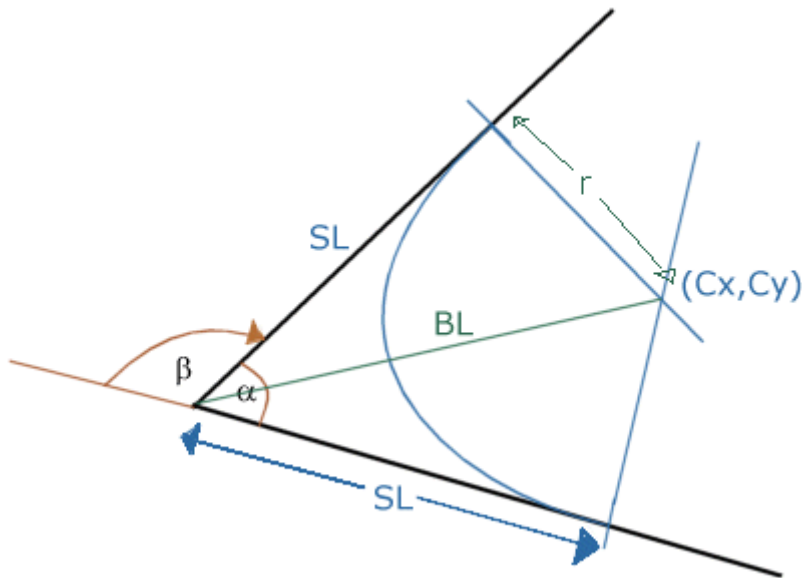
where a,b&c are the lengths. It doesn't matter how long each is, so using the I point & the two inner R points is the best idea.



So now you know the angle of the corner. The next stage is...

Stage II

Decide on the radius of each corner. This might be anything from 5 metres to 500 metres, depending on the sweepiness (made that word up) of the corner. E.g. La Source would be small radius, Curva Grande medium, and that bit between the 2nd and 3rd corners at Silverstone high (didn't know that bit was curved? he he). See the diagram below:



Right then. (Cx,Cy) is our goal here, the centre of the corner. Also, SL is a useful value, because it lets you work out the length of the straights (another goal).

$$SL = r / \tan (0.5*\alpha), \quad r = \text{radius of corner}$$

$$BL = r / \sin (0.5*\alpha)$$

Then, using the I point, BL , and α in relation to the angle of one of the segments (carry a starting known value right around the track), (Cx, Cy) can be found using simple trig.: $Cx = Ix + \cos (a) * BL$, $Cy = Iy + \sin (a) * BL$.

[Hang about, shouldn't we be using $a/2$? mmm...]

Beta = 180 - alpha. Beta is the curvature value needed for each corner.

The length of the corner, as the car travels is: $(\theta*\pi*\text{radius})/180$, where θ is the angle opposite α , $\theta=180-\alpha$.

Phew!

All of that should give you a big bunch of numbers on your nice pretty hand-drawn plan of the track you're creating. For each straight section, you should have:

- (Sx,Sy)
- Heading
- Length (not including the SL lengths at each end)
- (Ex,Ey)

For each curve section you should have:

- (Cx,Cy)
- (Sx,Sy) & StartAngle (angle from centre point to start point)
- (Ex,Ey) & EndAngle

- Radius
- Track Length (circumference of the arc between StartAngle & EndAngle)
- BL & SL for any later calculations/adjustments
- Curvature (Beta - negative for right hand turns, positive for left hand turns)

Next time I get around to writing something, I'll detail how to make the walls for the .TRK, i.e. the barriers, and different surfaces. Just look back at the detail above and remember that all that was just to get a centre line. Eek! **Grand Prix Legends .CAM File**

Format by P.A.Flack 1998

contact: phil@flack99.freeserve.co.uk
last revised 15/Dec/1998

Thanks to Edwin Solheim, the understanding of these files is almost complete!

Also, go see the [Car Camera](#) file format, by Brian Heiland

These files, of which there is one per track, define the locations of the two sets of cameras, TV1 & TV2 (along with the Pitlane camera). The header is very simple:

Bytes	Values	Notes
0-3	"MACT"	Section Type - TCAM
4-7	0x00000000	0
8-11	data size	Size of following data

The next four bytes give the number of cameras in the first set of cameras (TV1 first, then TV2, then pitlane)

Then, there are 56 bytes (14 floats/int) defining each camera.

Next, the number of cameras in the other set is given, then 56 bytes for each of those cameras.

Lastly, there are 52 bytes for what I assume is the pit-lane camera. This is a fixed view camera so it should help in the deciphering of the others.

Simple structure, but lots of numbers to work out:

What I've determined so far follows (small means numbers like 5.0, tiny means in the range of 0.1, miniscule means 0.001).

Number	Type	Description
1	float	Distance from s/f line in metres
2	float	Distance from track centre in metres
3	float	Elevation in metres
4	float	Distance camera takes over (-ve means the direction the cars come from)
5	int	Fixedness of camera: 0=follows cars, 1=fixed
6	float	tiny
7	float	tiny

8	float	tiny
9	float	Focus orientation
10	float	Tilt of camera in radians (pitch)
11	float	Tilt of camera in radians (yaw?)
12	float	some lens setting
13	int	0
14	int	0

Grand Prix Legends Car.CAM File Format by Brian Heiland for P.A.Flack 1998

contact: phil@flack99.freemove.co.uk
last revised 25/June/1999

These files, of which there is one per car, define the locations of six cameras per car, Nose, Gearbox, (Zeppelin or Shoulder), RollBar, Front Suspension & Rear Suspension. Zeppelin or shoulder view can be driven, but is not visible during replays. The header is very simple:

Bytes	Values	Notes
0-3	"MACC"	Section Type - CCAM (Car Camera)
4-7	0x00000000	0
8-11	data size	Size of following data

Each camera is defined by 36 bytes (9 floats).

There are a few values and units to work out yet, but most are found.

Formulas Radians to degrees = Radian * (180/pi)

Degrees to Radians = Degree * (pi/180)

Degree values are -180 to +180

Number	Type	Description
1	float	Distance from center point of car in metres (Neg=rearward, Pos=forward)
2	float	Distance from car center in metres (Neg=left, Pos=right)
3	float	Elevation in metres from car midpoint (Pos = Up Neg = down)
4	float	Yaw Angle of camera in Radians (- = Counterclockwise + = Clockwise)
5	float	Pitch Angle of camera in Radians (- = descend + = Climb)
6	float	Bank Angle of Camera in Radians (- bank left + = Bank right)
7	float	Wide Angle or Zoom (Unit Unknown Radian measure too? Greater value = wider angle)
8	float	Another Camera Yaw??? (Unknown)
9	float	Another Pitch Setting (Unknown)

Grand Prix Legends Setup File Format by P.A.Flack 1999

contact: phil@flack99.freemove.co.uk
last revised 11/Nov/1999

>> Thanks to **Brian Heiland**, there are now only 8 unknowns (the 800.0's and 7.0's) as opposed to 19.

The setup files are all the files within the players directory such as *.lo1, *.fe2, *.mu1, *.lo3, etc. They contain the settings for tire pressures, toe in angles, etc. I've only worked on the G1 settings, so there may be quite drastic changes between these and the G2 & G3 cars - most noticeable in the gears.

Bytes	Values	Notes
0-3	"PGTS"	File type, reversed to STGP (settings Grand Prix?) File sub-type, either 2 or 3 Size of following data (depends on sub-type)
4-7	2 or 3	
8-11	Data size	
12-	data...	

The rest of the file looks like this (Type: FL=float, Int=integer, Byte=8bit integer) (What? FL=Front left, FR=Front right, RL=Rear left, RR=Rear right).

The last four bytes only appear if the sub-type (bytes 4-7) is 3. If it is 2, the last four bytes don't exist. Strange...

Bytes	Type	What?
12-15	FL	Reverse Gear (-2.6923)
16-19	FL	Neutral Gear (0)
20-23	FL	1st Gear
24-27	FL	2nd Gear
28-31	FL	3rd Gear
32-35	FL	4th Gear
36-39	FL	5th Gear
40-43	FL	6th Gear
44-47	FL	Differential Ratio
48-51	Int	Reverse GearSet
52-55	Int	Neutral GearSet
56-59	Int	1st GearSet
60-63	Int	2nd GearSet
64-67	Int	3rd GearSet
68-71	Int	4th GearSet
72-75	Int	5th GearSet
76-79	Int	6th GearSet
80-83	Int	GearSet Differential
84-87	FL	Ramp angle (2nd value)
88-91	FL	Ramp angle (1st value)
92-95	FL	Tire pressure FL
96-99	FL	Tire pressure FR

100-103	FL	Tire pressure RL
104-107	FL	Tire pressure RR
108-111	FL	Wheel rate FL
112-115	FL	Wheel rate FR
116-119	FL	Wheel rate RL
120-123	FL	Wheel rate RR
124-127	Int	Bump FL
128-131	Int	Bump FR
132-135	Int	Bump RL
136-139	Int	Bump RR
140-143	Int	Rebound FL
144-147	Int	Rebound FR
148-151	Int	Rebound RL
152-155	Int	Rebound RR
156-159	FL	Camber FL
160-163	FL	Camber FR
164-167	FL	Camber RL
168-171	FL	Camber RR
172-175	FL	Unknown (800.0)(tire readings?)
176-179	FL	Unknown (800.0)(tire readings?)
180-184	FL	Unknown (800.0)(tire readings?)
184-187	FL	Unknown (800.0)(tire readings?)
188-191	FL	Unknown (7.0)(tire readings?)
192-195	FL	Unknown (7.0)(tire readings?)
196-199	FL	Unknown (7.0)(tire readings?)
200-203	FL	Unknown (7.0)(tire readings?)
204-207	FL	Bump rubber FL
208-211	FL	Bump rubber FR
212-215	FL	Bump rubber RL
216-219	FL	Bump rubber RR
220-223	FL	Toe in Front
224-227	FL	Toe in Rear
228-231	FL	Roll Bar Front
232-235	FL	Roll Bar Rear
236-239	FL	Ride Height Front
240-243	FL	Ride Height Rear
244-247	Int	No. of clutches
248-251	FL	Gallons of fuel
252-255	FL	Steering Ratio (n:1)
256-259	Int	Front Brake Bias
260	Byte	Asymmetrical Edit (0= tick in game, 1=cross)
261	Byte	0x20
262	Byte	0x20
263	Byte	0x20

And now for the gearsets:

ID	Values	Calculated
0	13/46	3.5385
1	14/46	3.2857
2	14/45	3.2143
3	12/35	2.9167

4	13/35	2.6923
5	16/42	2.625
6	15/38	2.5333
7	14/34	2.4286
8	14/33	2.3571
9	14/32	2.2857
---	-----	-----
10	16/42	2.625
11	15/38	2.5333
12	14/34	2.4286
13	17/41	2.4118
14	14/33	2.3571
15	14/32	2.2857
16	15/33	2.2
17	16/31	1.9375
---	-----	-----
18	15/32	2.1333
19	15/31	2.0667
20	16/32	2
21	16/30	1.875
22	17/31	1.8235
23	17/30	1.7647
24	17/29	1.7059
---	-----	-----
25	14/34	2.4286
26	17/41	2.4118
27	14/33	2.3571
28	14/32	2.2857
29	15/33	2.2
30	15/32	2.1333
31	15/31	2.0667
32	16/32	2
33	16/31	1.9375
34	20/38	1.9
35	16/30	1.875
36	17/31	1.8235
37	17/30	1.7647
38	17/29	1.7059
39	20/34	1.7
40	18/30	1.6667
41	18/29	1.6111
42	18/28	1.5556
43	19/29	1.5263
44	21/32	1.5238
45	19/28	1.4737
46	18/26	1.7647 ?
47	19/27	1.7059 ?
48	24/34	2.4286 ?
49	20/28	1.4
50	21/29	1.3810
51	24/33	1.375

52	25/34	1.36
53	20/27	1.35
54	24/32	1.3333
55	25/33	1.32
56	19/25	1.3158
57	23/30	1.3043
58	21/27	1.2857
59	25/32	1.28
60	26/33	1.2692
61	19/24	1.2632
62	23/29	1.2609
63	25/31	1.24
64	21/26	1.2381
65	26/32	1.2308
66	19/23	1.2105
67	24/29	1.2083
68	27/32	1.1852
69	22/26	1.1818
70	25/29	1.16
71	19/22	1.1579
72	27/31	1.1481
73	28/32	1.1429
74	22/25	1.1364
75	25/28	1.12
76	27/30	1.1111
77	28/31	1.1071
78	29/32	1.1034
79	22/24	1.0909
80	24/26	1.0833
81	25/27	1.08
82	23/24	1.0435
83	24/24	1
84	25/24	1.0417
85	26/24	1.0833
---	-----	-----
86	7/31	4.4286
87	8/31	3.875
88	9/31	3.4444
89	10/31	3.1

Last 4 are differentials. Not sure about the few out of sequence values in the middle (nos. 46-48)

Grand Prix Legends controls.cfg File Format by P.A.Flack 1998

(initial info. by [Marcel Borsboom](#))

contact: phil@flack99.freerve.co.uk
last revised 11/11/1998

The controls.cfg store the controller settings for each user of the game. I haven't really looked at this yet but the following is offsets in the file for the keys (assuming char data type):

Bytes	Values	Notes
0x02d9	key	Throttle
0x02f6	key	Brake
0x0313	key	Steer Left
0x0330	key	Steer Right
0x0387	key	Shift Up
0x03a4	key	Shift Down
0x04a9	key	Pit Board
0x0500	key	Raise Arm
0x06d0	key	Look Left
0x06ed	key	Look Right

GPL Track Creating Downloads

You'll be wanting some goodies then ? Here you go then- just make sure you read the readme's before asking any questions !

Anything you find here is property of the author(s) shown; you must obtain permission from them before distributing their work, making money/profit from their work, or in any other way utilising their work for a purpose other than creating freely distributable tracks for Grand Prix Legends.

Please note that some programs are not held here, so you will be taken to another website.

Applications

Program	Brief Description	Download / Link	Author(s)
trk23dow 27-Jan-2001	converts GTK -> 3DO (essential)	[Download] 116kb	Phil Flack
GPLTrk 2.1	creates GTKs/TRKs (essential) - no longer supported by Peter	[Download] 279kb	Peter Prochazka
MipMan (beta!)	creates 16bit MIPs (and 4bit non-transparent)	[Download] 28kb	Phil Flack
SrbMan (beta!)	creates SRB+3DO from mip	[Download] 11kb	Phil Flack
ASE 2 3DO	converts ASE (3DSMax) to 3DO objects	[Download] 44kb	Phil Flack
Sciss 14-Nov-2000	checks for missing files	[Download] 26kb	Phil Flack
WinMip 2	creates MIPs, PBFs, ...	link	Klaus Hörbrand
txt2gtk	so Excel can be used for altitudes	[Download] 10kb	Phil Flack & Peter Prochazka
GPL Track Cam	create cameras for your track	link	Joachim Blum
rpy2lp	converts replay to LP (AI) files	link	Nigel Pattinson

LP-Edit	so you can tidy up the LP (AI) files	[Download]	François Dubuc
RaceCon	see tracks outside GPL, check for errors	link	Nigel Pattinson
dooDAT	dat packing/unpacking	link	double D Software
GPLDFM	dat packing/unpacking	[Download] 42kb	Phil Flack
GPLTrackEditor	create trackside objects (3DOs)	[Download]	Paul Hoad/Phil Flack
GPLDump	Replay analyser, groove data extractor	link	Juha Kallioinen
GPLReplayAnalyser	Replay analyser, groove data extractor, does the dishes, walks the dog...	link	Jonas Matton & Martin Granberg

Example Files

File	Brief Description	Download / Link	Author(s)
MyTrack	Files for the My First Track tutorial	[Download] 102kb	Matt Knutsen
Altitude Spreadsheet	For use with txt2gtk; altitudes	[Download] 434kb	Martijn Keizer

Links

Much as we try, we can't provide everything ;-)
 These are off-site links, the content of which we have no control over.
 Think your website is worthy? webmaster@gplea.org

www.theuspits.com/	You'll find wonderful things here, including Nigel Pattinsons programs (RaceCon, RPY2LP, etc.)
eagle woman	Alison Hine's great GPL site
www.simracingmag.com/	GPL, WSC, news, tips, handy GPLEA mirror site :)
www.papy.com/	Papyrus. They made something called GPL
www.bigfoot.com/~GPL_Replay_Analyser/	Home of GPL Replay Analyser (get it ! get it now !)
http://www.horbra.de/winmip/	Home of WinMip
http://www.gplea.org/	Obligatory link to the best GPL page around ;)
Ultimate GPL Linkspage	Can't find a GPL-related item? If it ain't linked on this site, it ain't... erm :)
Track Reverser	Bored of Indigo-syndrome? Need to practice Wrong Way?

Repository

There is not much here, don't get too excited...
Can't make it yourself ? Not enough hours in the day ? Never fear, there's plenty here !

Remember!
If you use something from the Repository, please give credit to the creator(s) of the files you use,
preferably drop them an e-mail of thanks as well.

Objects

Categories

Trees, bushes, people, lights, cars, you'll find it here (unless it hasn't been made yet...)

Trees, bushes, etc.

People - single/groups

Lights, PA poles, flags...



x objects

Grandstands



Pit-lane buildings

Signs/Billboards



456 signs

Other buildings

Vehicles

Miscellaneous

Textures

Categories

Grass, bricks, dirt, sky, rabbits, you'll soon get a retro feel to your track with this compilation of goodies... Click on the icons to get the big picture.

Ground
Solid Walls
Trees and other green things
People & Wildlife
Adverts



by Matt Knutsen:
Miscellaneous

Optimizing FPS

The Golden 36 FPS

Without 36 frames per second, GPL can be a very difficult beast to master (if it wasn't already ;-). In order to achieve 36FPS on low-end systems, you should consider everything here:

4bit vs. 16bit

While 16bit textures look so much nicer than 4bit textures, they also occupy more texture ram, and on older graphics cards/software rendering, they slow things down quite significantly. ***Wherever* possible, use 4bit textures.** Another point to consider is the total size of your textures - if you use a different texture for every house, dozens of grass textures etc., you could easily have more than 4 or 8mb of textures. A Voodoo2 12mb graphics card (old technology!) has 8mb of texture ram - if you exceed 8mb, then the textures will have to be swapped in and out of system ram, causing a bottleneck.

Start/Finish Line

With 20 cars and all their associated physics & graphics, you don't want an over-complicated track adding to the CPU load. Keep the start/finish area simple - only use a few TRK walls, avoid pit garage buildings with individually modelled flag poles, air hoses, etc.

TRK Complexity pt I - Walls

You can usually have as many different walls as you like (up to a reasonable limit - 20 or so). What you should try and avoid, however, is vertical walls - armco, hedges, etc. These add to the CPU load much more than ground-level walls - 3 times as many polygons, as well as additional physics calculations. In FPS-heavy areas of the track you should consider removing some walls.

Whatever you do, don't have walls of zero-thickness, and don't 'layer' walls to make a stepped wall going back from the track. You can do this occasionally, but a 20 step embankment would be silly!

TRK Complexity pt II - Sections

Straight sections are not a problem, Corner sections are. When you create a single corner section, it seems just like a bent straight section. It is actually created from ***at least*** 4 sections, and depending on your <trackname>.set Corner_Error value, it will be created from 4, 8, 12, 16, etc. sections. This is not good !

If you have a blocky looking corner, consider increasing the number of polygons using Unknown4 flags, not changing the Corner_Error value. [Unknown4 + 65536, 2048, 4096 are the flags to use]. Although the additional polygons will slow things down, they do not slow things down as much as the extra sections. A very good example of this is a chicane, where you typically have 4 corner sections

and some straight sections. When you look at that part of the track when driving, you are looking through a large number of sections, all at different angles. GPL does not like doing this !

Polygon Count pt I - Track

More polygons = better looking track, worse FPS.
Fewer polygons = worse looking track, better FPS.

You can decrease the number of polygons using the Unknown4 flags and <trackname>.set. Ideally you want to start with a very low polygon-count track and gradually increase the number of polygons until everything look acceptable.

Polygon Count pt II - Objects

In general, trackside objects are much better in terms of polygon count than the track. You should only see FPS hits if you add very large (in terms of dimensions) objects, or if you add a lot of objects (e.g. 100 trees down one straight).

Polygon Overlapping

Drive around Crystal Palace and you will see a lot of large trees down each side of the track (in places). A lot of these trees overlap each other, so the same area of screen is being filled over and over again. This is not good, but often unavoidable.

Racing Groove

The racing groove, when turned off, can boost FPS by 33% on low-end systems (thanks Niels!). Do you need racing groove around the entire track ? If not, set the transparency level in <trackname>.GRV to 0 - trk23dow will not create any polygons then. Also, there are some Unknown4 flags that will let you reduce the number of groove polygons (flags 8192, 16384).

Existing Tracks / Tracks In Production

Making a decent track for GPL takes a long time, so it makes sense not to repeat what someone else has already created. We will update this page with information on tracks that have been finished and tracks that have been started. If you are seriously making a track, please let us know so everyone else can see. You can be listed anonymously if you prefer to work in peace. email: webmaster@gplea.org

Existing Tracks

Here is a list of tracks that you can currently obtain for Grand Prix Legends. We hope it is complete - if you know otherwise, please send us an email at webmaster@gplea.org

Reversed tracks not included - You can get the Track Reverser program from here:

http://members.nbci.com/_XMCM/pez805/index.html

Converted tracks not included.

Track Name	Year, Notes	Author(s)	URL
Kyalami	1967	Papyrus	GPL Original
Mexico	1967	Papyrus	GPL Original
Monaco	1967	Papyrus	GPL Original
Monza	1967	Papyrus	GPL Original
Mosport	1967	Papyrus	GPL Original
Nürburgring	1967	Papyrus	GPL Original
Rouen-les-Essarts	1968	Papyrus	GPL Original
Silverstone	1967	Papyrus	GPL Original

Spa Francorchamps	1967	Papyrus	GPL Original
Watkins Glen	1967	Papyrus	GPL Original
Zandvoort	1967	Papyrus	GPL Original
Table 2: 3rd Party Addons, Real Life			
Track Name	Year, Notes	Author(s)	URL
Anderstorp	1973ish	Martin Granberg, Jonas Matton, ...	http://w1.314.telia.com/~u31413736/
AVUS		Scott Pryzbylski	http://www.fortunecity.com/...
Brands Hatch		David Noonan	http://www.theuspits.com/
Bremgarten	1954	Sim Racing Club Bern	http://drive.to/srcb/
Bugatti au Mans	1967	Roland Ehnström, ...	http://home.swipnet.se/~w-36707/bugatti/
Crystal Palace		GPLEA	http://www.gplea.org/
Dubai		David Noonan	http://www.theuspits.com/
Goodwood	1999-Revival	GPLEA	http://www.gplea.org/
Imola		David Noonan	http://www.theuspits.com/
Österreichring		David Noonan	http://www.theuspits.com/
Road Atlanta		David Noonan	http://www.theuspits.com/
Snetterton	1964	GPLEA	http://www.gplea.org/
Snetterton	1967 (new Russell)	GPLEA	http://www.gplea.org/
Solitude		Martijn Keizer, GPLEA	http://www.gplea.org/
Table 3: Fantasy Tracks			
Track Name	Year, Notes	Author(s)	URL
Glacier		Mark Beckman	
Pants Ring	n/a	GPLEA	http://www.gplea.org/

Tracks In Production

Here is the list of tracks as of 18-Feb-2001.

Table 5: Real Life Tracks			
Track Name	Year, Version, Notes	Author(s)	URL
Clermont Ferrand	1965-1972	GPLEA	http://www.gplea.org/
Dundrod	1954	Shaun McGoldrick	
East London			
Keimola	1970	Greger Huttu, Olli Suominen	http://turn.to/Keimola/
Hockenheim	1967	GPLEA	http://www.gplea.org/

Knockhill			
Le Mans 24hr	1967	Anonymous	
Magny Cours	post 1993	Martin Goedhart	
Montjuich	1969	Olli Suominen, D-C.Bermudez, G. Huttu	
Mont Tremblant	1970	um... who? Eric Bourgouin?	
Monza Banked		Anonymous	
Norising		Lou Magyar	
Oulton Park	1960s	Anonymous	
Reims			
Warwick Farm	1967	The Tasman Project	http://go.to/tasmanproject
Wigram	1967	The Tasman Project	http://go.to/tasmanproject
Table 6: Fantasy Tracks			
Track Name	Year, Version, Notes	Author(s)	URL
Ed-Ring		GPLEA	http://www.gplea.org/
JTRC2		GPLEA	http://www.gplea.org/

GPL Track Creating

Thanks to all...

[in alphabetical order]

Peter Allam
Jay Beckwith
Joachim Blum
Daniel Cean-Bermudez
Tony Churly
Francois Dubuc
Roland Ehnström
Brian Fox
Ray Geering
Jon van Ginneken
Martin Granberg
Brian Heiland
Niels Heusinkveld
Alison Hine
Nate Hine
Paul Hoad
Lawrence Holbert
Klaus Horbrand
Greger Huttu
Pedro Johansson
Bruce Johnson
Vinny Kwiker

Joe Lafrite
Al Layton
Bart-Willem van Lith
Greg Longfield
Juha Kallioinen
Lou Magyar
David Mocnay
Mat Mann
Saro Marcarian
Shaun McGoldrick
Remco Moedt
David Noonan
Ron O'Dell
Rick Osborn
Papyrus/Sierra
Nigel Pattinson
Peter Prochazka
Scott Pryzbylski
Jamie Rosenzweig
Thorsten Rueter
Simone Sorrentino
Ben Summers
Olli Suominen
Gerald Swan
Bill Tillman
Dirk Wagner
Chris & Tony West (go WSC!)
Tim Wheatley
Martin Zutter

...your tools, programs, feedback, game (!) have made this possible.

[apologies to anyone we missed!]

This Track Creating Guide was brought to you with the letters G, P and L, and the number 1967. These people also did a bit of typing & doodling, mostly when they should have been working:

Phil Flack
Ed Solheim
Matt Knutsen
Martijn Keizer