Landwirt objects introduction. Revision 0.01, Dec 23 2013.

Topics:

1. Introduction

This documents briefly covers topics concerning creation and modding of Landwirt objects, esp. Vehicles. This is not a full-blown instruction and don't blame me if you alter some files and something goes wrong. The game and engine are well-tested, but not bullet-proof. Always make backups.

Several topics may not be covered. Hopefully they will be sometime soon.

2. General do's and dont's

- Do not use language specific characters in file names/paths – the game will reject these
- Do not use spaces in file names/paths – the game will reject these as well. Sometimes filenames are used as parameters and they have to be a non-interrupted string.
- When the game crashes, always check the !console.txt and !error.txt (if exists). Debug information and error messages may be stored there.
- If a mesh is not found or corrupted the game will not exit (unless due to DirectX crash), but inform about the error in !console.txt instead.
- If a texture is not found, the game will exit immidiately, also with error message stored.

3. Meshes

1. What is a mesh?

Mesh is a set of textured triangles, non-animated, imported to the game. Displayed, it creates most of the visual game content. It's just a static object.

Meshes may have a lot of specific features – different physics params, action switches, may be trees, water – you name it – but these are not covered in this document.

Different textures may be used on a mesh, however it's always better to use fewer textures than many. Switching textures is render cost-effective, so it's better to group stuff on one large texture, than split them.

On the other hand textures are also materials, so it is better do have metals on one texture, glasses on the other, tires on yet another and so forth. Just don't use different texture for every door knob, or soon you'll face 5 FPS.

2. Creation/Exporting

Meshes are created in any suitable 3D software of your choice that exports to .x (Microsoft DirectX format). If your software doesn't export to .x, chances are there is a suitable exporter like Panda.

Using Panda as an example, here's what should and what should not be included while exporting a mesh:

Include mesh definition, materials, mesh normals and mapping coordinates. Do not include animation nor bones. Do not optimize mesh.

No animation is required.

Do NOT convert texture maps, do not include effect files.

DX File type should (and this is a strong SHOULD) be text, coordinate system is Left Handed Axis.

All parts of exported mesh must be properly textured – no empty (not textured) subsets are allowed, or the mesh will not load. Also it must be a textured geometry only – no cameras, lights or other objects.

Game unit = one meter.

Vehicles must be exported facing north, standing on the zero point. In other words, the zero point has

to be in the middle of the car, just below it – the vehicle stands on the 0 surface.

All the moving parts of the vehicle (wheels, mudguards, pistons, attachments) have to be separate meshes, exported with their 0 points in rotation pivot.

When creating/exporting meshes, pay attention to normals as they define culling.

32-bit meshes are supported, but they are less effective than 16-bit meshes (memory usage). Use meshes below 32k verices/triangles if possible. Most of our machines are above this point, but most of the world objects are below.

3. File structure

Meshes are stored in the prefabs subfolder of the game. Specifically vehicles are in prefabs\pojazdy, each vehicle (and parts) in it's own subfolder. So for instance a main combine component and all its parts will be in prefabs\pojazdy\combine01_a\combine01_a_nocenter.x

Some components may be shared (several wheels, shafts), so they may be in some other machine's directory. Examine vehicle scripts.

For meshes the path is important, so when describing vehicle script or sockets it is always included.

Now for the hard part – introducing !opts.

We recommend meshes (prefabs) to be kept as text files, as it is much easier to look at them, debug (see if there are empty subsets, or weird textures, or whatever) and just externally edit, for instance to copy an object and use different texture on it.

That said, text files are large, unoptimized and take long to read. So we convert them to !opt files and that's what the game uses in the end. Opts are specialised copies of respective prefabs and they are stored in the !opt folder (storing full prefab path/filename in !opt filename). !opts are optimised, with tangents and binormals precalculated and in proper vertex format – so they are very fast to read.

The order is like this:
Game wants to load a mesh, so it searches for the !opt. If the !opt is found, it is read, end of story. If the !opt is not found, the game reads a text .x from prefabs, converts it to !opt, stores in the !opt directory. Next time the game is run, the !opt is read.

So – if you are exporting a mesh that already exists, it is not enough to just overwrite the prefab. You have to remember to delete the !opt, or it will be read instead (and it contains older version of the file). This may be confusing, so if you are making many changes and exports and want to test them, you may disable this mechanism by editing ar.cfg and setting the UseOpt to 0. This will make the game load longer but use prefabs only, so you will see all the exported changes with no need to delete !opts every time.

Do NOT delete any other !opts, specifically do not delete any !opt that is not a vehicle. Prefabs for them are not included in the game and you will lose objects from the game.

4. Center vs nocenter.

It is a common routine when creating a set of objects (like buildings for a city), to create them side by side in one 3D edited file (to compare sizes, to see how they will look together, to have less files to edit – multiple reasons) and then export one by one to have a set of separate buildings to use in the game. Such objects will store their positions, so when inserted in the game's (0,0,0) point, they will be transposed as they were in 3D software – and we want them to be in the point where we want them, not transposed as they were while editing.

To have such objects easy to use, the game automatically moves each mesh so that it's local zero point is in the exact middle of it's bounding box. This is very useful – you export the object from 3D software from anywhere it is, and it is put exactly in the cursor position in editor when you add it into the game. This is the default setting.

However it is also pretty common for several objects that you want them positioned precisely as you set them in 3D software and you don't want the game to move them. All the attachable objects for instance – wheels, doors, attached machinery parts – mostly you want the rotation pivot in the zero point and it should not be transposed. If this is the case, simpy add _nocenter in the end of prefab file name and the game will not try to move it. Note, it must be not anywhere in the name, it's got to be in the end, like vehicle01_a_nocenter.x

If you take a look at the prefabs\pojazdy subfolder files, quite a lot of these have the _nocenter in the end. It's because typical vehicle is a hierarchy of attached objects and each of them has a specific place to connect to.

5. Sockets

Sockets are specific, named points put somewhere on the vehicle. They are like markers – specific points that define something (mostly connection points) on the mesh. Say you have car chassis and you want rotating wheels connected – you create a wheelless car, 4 separate wheels (or preferably one

left and one right wheel, each to be used front and rear), define 4 sockets in the chassis, each for the wheel. The wheels should be exported _nocenter, so that their zero point fits the socket in the car.

The whole idea is simple, but it usually takes some planning on more complicated machines, when the hierarchy is complicated and attachments are attached to other attachments.

Sockets are defined and managed in the game editor, but they also can be peeked at in the scripts\mesh_sockets folder. Each socket is defined by a unique (per mesh) name and 3 numbers describing translation from a mesh 0 point.

6. Misc

Questions?

4. Textures

I am assuming you know the basics, so I will not be covering them here. This will cover mostly Landwirt-related topics instead.

Textures must be power-of-2 sized (remember the no-space-nor-weird-characters-in-filenames rule).

1. Types

Functionally speaking, there are 2 types of textures in the game – flat 2D and cube textures, with 2D textures being the majority. There is also a number of render targets, but since they are not externally accessible, they will not be discussed here. For most of the purposes, you will be dealing with typical 2D textures, with alpha masks or not.

Game uses 2 types of files – Targa (.tga) and .dds. The latter are mostly better as they are smaller, come with mipmaps already and read faster. Sometimes in rare cases their compression shows up, but it's still better to repaint the dds than use big .tga. It is still possible though. We use .tga files mostly for HUD and skies, as these require highest quality possible.

2. Folders/files

There is one big thing you always have to consider when using textures in Landwirt: they all have to have unique names. Our textures do NOT use paths nor are identified by a path. Simply different exporters from different 3D software either do or don't include path to a texture – it was always a mess and we gave up on it. So it is not good to have 2 different diffuse.dds in two different directories – you will never know which one will be used.

Mesh textures are stored in textures directory, sprites in sprites, animated (skinmesh) textures in skins – but it's just for clarity. Basically a texture can be stored in any subfolder, and as long as it's name is unique, it will be found and used.

Same distinction comes for tga versus dds – you may use both formats and the game will try to find whichever you use. If you texture the mesh with a tga, and then save the texture as dds, it is fine - the game will find and use the dds instead of tga. There's no need to remap the mesh with dds if you convert the texture. Just don't use both the diffuse.dds and diffuse.tga in game folders, as the game will choose one of them and you can't be sure which one. This may be especially confusing if you use tga with a specific name in one folder and dds with the same name in another – be mindful while naming the textures!

It's a good practice to start texture name with unique identifier – all of our hud textures start with hud_, sprites start with spr_, vehicle textures start with vehicle name and so forth. This is not a requirement, but will save a lot of debugging time.

Also it is always better to have as few textures as possible, so some of our vehicles share some parts – pistons, wheels, window panes etc.

3. Normals

Due to light rendering requirements, all the textures in Landwirt must have normals (ask google if you need info on what a normal texture is). All the above requirements apply to normal textures as well – they have to have unique names, power-of-2s, no spaces in filenames, either tga or dds and so forth.

By default normal for a texture is identified by a _normal added to the end of the filename, so "combine01_a_diffuse.tga" will look for a "combine01_a_diffuse_normal.tga" anywhere. It will NOT find "combine01_a_normal.tga". But it will also find and use "combine01_a_diffuse_normal.dds" obviously.

It is possible to assign different normal to a texture – typically done from Texture Editor, but can also be defined directly in a texture script (scripts\textures). For instance if you use different paintings on the same machine, they may share the same normal to preserve VRAM.

Normals do not have to be the same size as base texture. They also may be of different type (tga vs dds). They also don't have be in the same directory as the base texture – paths are ignored for all the textures.

4. Transparency, specular, environmental.

There are a lot of surface effects, some obvious, some subtle. They are assigned in the Texture Editor.

However even without this tool, one may manipulate them through scripts (scripts\textures). Here's brief overview of some of them:

- Transparency is controlled via alpha mask of diffuse texture. It is 2 states only – below 0.5 it is transparent, while above 0.5 it is opaque.
- Translucency is fluent, in whole range of alpha mask.
- Detail is another, dense texture multiplied over the base texture. It can be also another normal (detail_as_normal), and in this mode it details normal and lighting effects. It can also multiply main specular mask. Controlled via alpha.
- Environmental is environmental reflection with a cube texture, controlled via alpha.
- Specular is light specular reflection, controlled via alpha and 2 parameters.
- Other options may be deduced from their names in texture script.

5. Vehicle scripts

Each machine is described and controlled by a proper script located in scripts\vehicles. It may not be fully readable, but there are plans to translate it whole to English (right not several commands are in Polish). It should be mostly obvious though.

6. Example

Here's what I would do if I wanted to create another vehicle (perhaps textured differently) and put it into shop. Let's take for example tractor05_D. Here's the steps:

- Copy prefab folder to another name – prefabs\pojazdy\tractor05_D to (for instance) prefabs\pojazdy\tractor05_custom
- may change the main prefab name to fit the folder name, may not – it's for clarity. Let's rename prefabs\pojazdy\tractor05_D\tractor05_D_nocenter.x to prefabs\pojazdy\tractor05_custom\tractor05_custom_nocenter.x to match folder name. If you want LOD file to work too, you need to rename the LOD_tractor05_D_nocenter.x to LOD_tractor05_custom_nocenter.X (to match the main file).
- If you are not changing wheels, mudguards and so forth, you may delete these from new folder – script controls where they come from, so they may as well be taken from the original model.
- Now, the main texture the model uses is tractor05.dds (textures\pojazdy\tractor05_D). Copy it to tractor05_custom.dds. Paint pink, or whatever you like.
- Edit prefabs\pojazdy\tractor05_custom\tractor05_custom_nocenter.x – notepad will do great. Find reference to tractor05.dds and change it to tractor05_custom.dds. Save file.
- Copy scripts\textures\tractor05.txt to scripts\textures\tractor05_custom.txt. Edit tractor05_custom.txt and change texture name to tractor05_custom.dds. This copies all the descriptions from old texture to new one. You may play with the settings of course.
- Copy scripts\mesh_sockets\prefabs_pojazdy_tractor05_D_tractor05_D_nocenter_x.txt to scripts\mesh_sockets\prefabs_pojazdy_tractor05_custom_tractor05_custom_nocenter_x.txt to

copy all sockets from old tractor to new. Edit the file and change mesh_filename (first line) to reflect new path and filename as well.

- Now for the vehicle script. Copy scripts\vehicles\tractor05_D.txt to scripts\vehicles\tractor05_custom.txt and edit the file, so that prefab (2nd line) shows proper path and filename.
- You're good to go – you have new tractor. You may play with it's script now. Keep in mind it's not bullet proof though.

Additional changes may be creating new shop thumbnail (copy misc\thumbnails\th_tractor05_D.dds to misc\thumbnails\th_tractor05_custom.dds and put that name into the "icon" command in the vehicle script.

Also script name is the "Vehicle" command in the script. Proceeded with ">" it means the name is in localisation scripts – you may delete the ">" and just put your full name like this:
Vehicle My own badass tractor!

This may look complicated, but in reality it's the same operation on a bunch of scripts (copying and altering names) – no big deal. Same should work if you wanted to create your own tractor for instance. A bit of work on sockets and other parts and all should be great.

Have fun!

7. Conclusion

This doc is brief and it is by no means a full blown documentation. Use it and have fun – or don't. I take no responsibility for wrong use (especially if you spoil your game files), but wish best of luck to modders – you are very creative, gals and guys, and it's always great to see what you come up with.