Creating 3D models for Theatre of War

Draft version 0.2 eng

By Knokke

This tutorial will attempt to give you a better understanding of the inner working of Theatre of War and Theatre of War 2, and the way of adding new 3D models to the game. This tutorial is by no means perfect, as it is only my understanding for the time being of the procedures to follow, and my knowledge is somewhat limited, seing that i'm not able yet to produce 100% of the time a model that will work. Creating a model for ToW is a complex process, with many possibilities of doing some mistake that would prevent the model to work in game. Adding a new unit to ToW has been for me a process of trial and error until I managed to have something working.
Any addition or correction to this tutorial will be greatly appreciated.

The tutorial intended to be divided in two parts: first part would have dealt with creating new models and set them up for import into ToW with MshConverter. The second part would have dealt with how to setup a 3d model so that it could be imported into ToW with 1C proprietary convertion tools. But now that 1C has publicly released their tools with the documentation, the second part of this tutorial is no longer needed. The first part about using MshConverter is also of less interest now, as it would be more appropriate to use 1C tools to handle the convertions.

So probably only the chapter about the configuration files would be of any interest to a modder (and theses aspects have already been covered in Oudy's « ToW Modding FAQ 1.02 » as well as in 1C paper « Unit modeling guide for Theater of War V.1.0 eng » and the latest « Creating new units in Theatre of war » released with theToW Mega Mod Pack.

Tools needed to create new models for Theatre of War:

First, there are a few tools that you'll need to create your models:

_SFSExtractor for ToW (with the filelist.txt corresponding to the game version that you'll work for)
SFSExtractor is an utility that allows you to extract the game files from the compressed SFS files, so that you can have a look at them, and modify them. I'll not discuss in detail how to extract the files from the SFS, as it is explained within the documentation included with SFSExtractor.

_A 3D modelling package to create your 3D model. 3DSmax would be the obvious choice, since you have to export your model to *.3DS format to be able to convert it to *.msh format in MshConverter. Any other 3D package could do the job, since most of them can export your model to a format readable by Max. MshConverter can also read a few other formats, but since i've not tested them, I'll assume you're using 3DSmax, as it is the most straightforward way of adding a model to the game.
Note for those who can't afford an expensive 3D package like Max: there might be the possibility to use Blender3D (a free 3D application that is able to import/export 3DS format). I didn't test it yet, but at least you can load in it a 3DS file extracted from ToW via MshConverter. There might be

some trouble to find the right scale for the scene, and perhaps with materials but it is worth investigating further.

_You'll need MshConverter v1.18 by Dr Jones (the most up to date version at the time i'm wrinting this tutorial). This utility allows you to convert the 3DS file you created in Max to ToW format: *.msh.

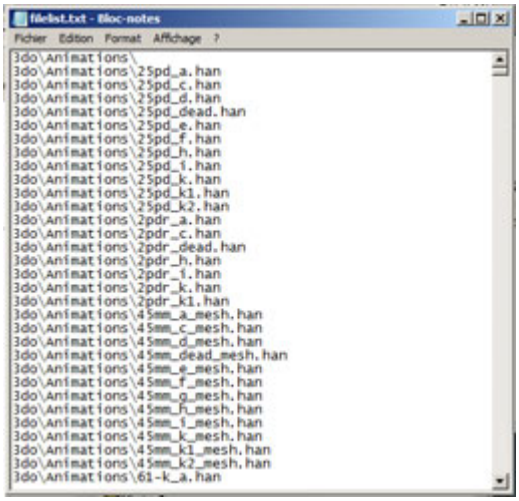
_A graphical utility like Photoshop or the Gimp to create your textures. I assume you'll be using Photoshop because you'll need the Nvidia plug-in for reading and saving *.DDS files, and creating the normal map textures for your objects.

_You'll need to download and install the Nvidia Photoshop plug-in for opening and saving *. DDS files (you can find it on the Nvidia website).
http://developer.nvidia.com/object/photoshop_dds_plugins.html

_You'll also need a few reference papers and 3D files dating from ToW1 that will prove very useful:
ToW Modding FAQ 1.02 by Oudy (available in Battlefront's Repository)
Unit modeling guide for Theater of War V.1.0 eng (by 1C dev. team)
3DSmax models of the M4 and M6 setup for import in 1C converter (by 1C dev. Team)
The Max files provided for the many models with the ToW Mega Mod Pack will be an excellent reference material too. (available in Battlefront's Repository)

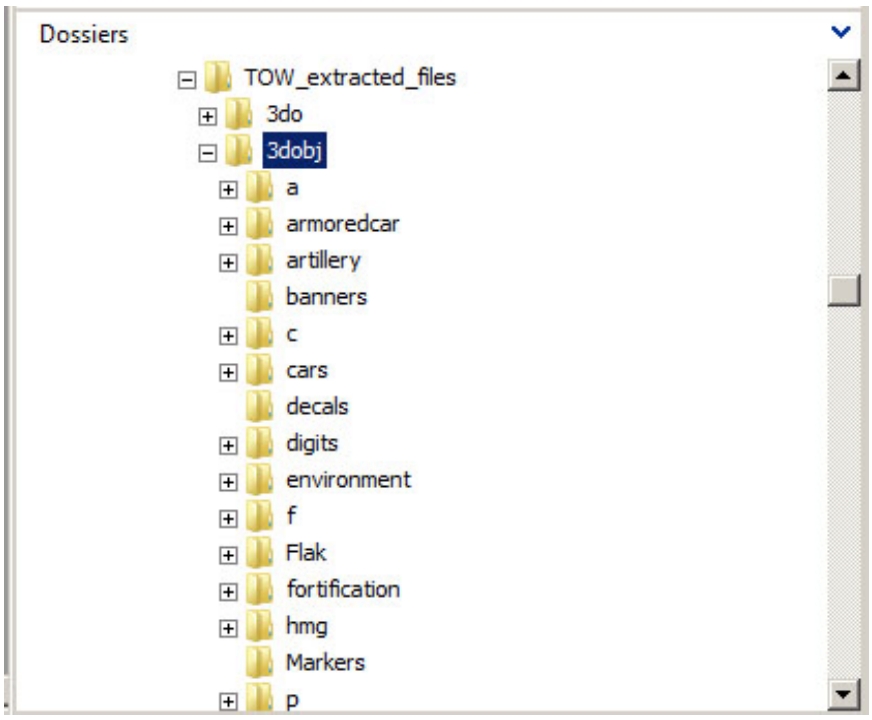
Basic understanding of the organisation of files in ToW:

Basic understanding of the way ToW handles files is needed, because you'll have to recreate files following the exact same structure as ToW is using.
All files that you can modify in the game are archived inside SFS files, like 3dobj.SFS, Maps.SFS, Data.SFS... For accessing theses files for reference or modification, you need to extract them with SFSExtractor and the corresponding filelist.txt: the text file is a list of all the so far know files contained in the SFS archives with their path.



^ The filelist.txt

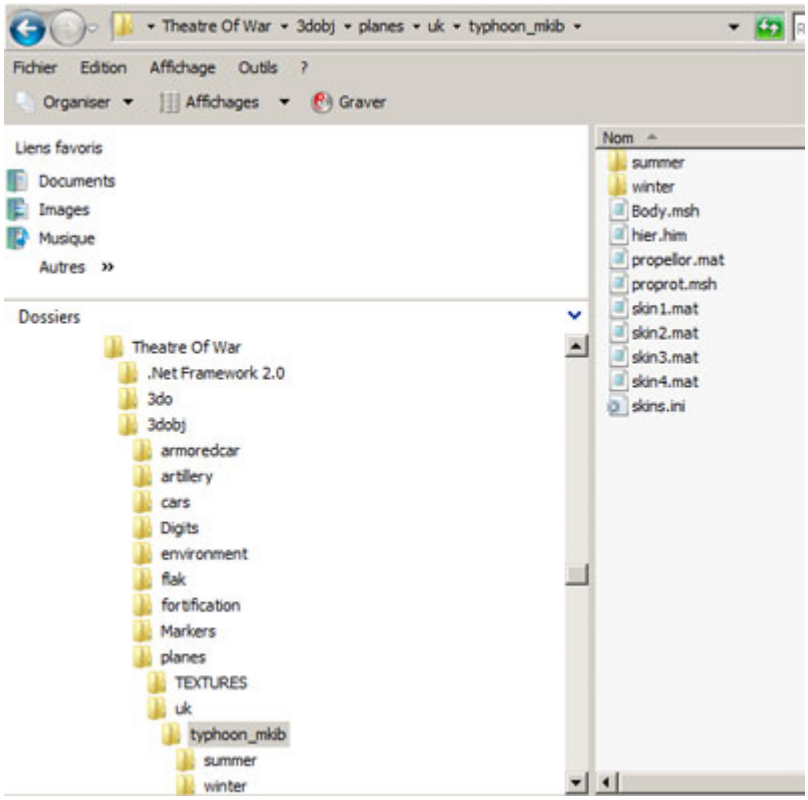
Theses files have to be extracted to the place of your choice, but they should not be extracted inside the main ToW folder. You can extract them in a ToW_Extracted_files folder somewhere on your hard drive, for example.



^ The content of the SFS files extracted to a folder

Once the content of the SFS files has been extracted, you'll find a lot of directories inside, and a lot of files of various type inside the directories. They are the texture files, the 3D model files, the sound files and the configuration files for the game. The game loads theses files from the SFS first. But if you create a folder containing files corresponding to a file located in a folder within the SFS and put this directory and file inside the ToW main folder, the game will load this file instead of the one inside the SFS file (the modded files inside the Tow main folder will have priority over the ones inside the SFS files).

For creating new files for the game, it is the same process: you have to create the file stucture inside the main ToW folder according to the way it is organised in the SFS compressed folder for a similar file:



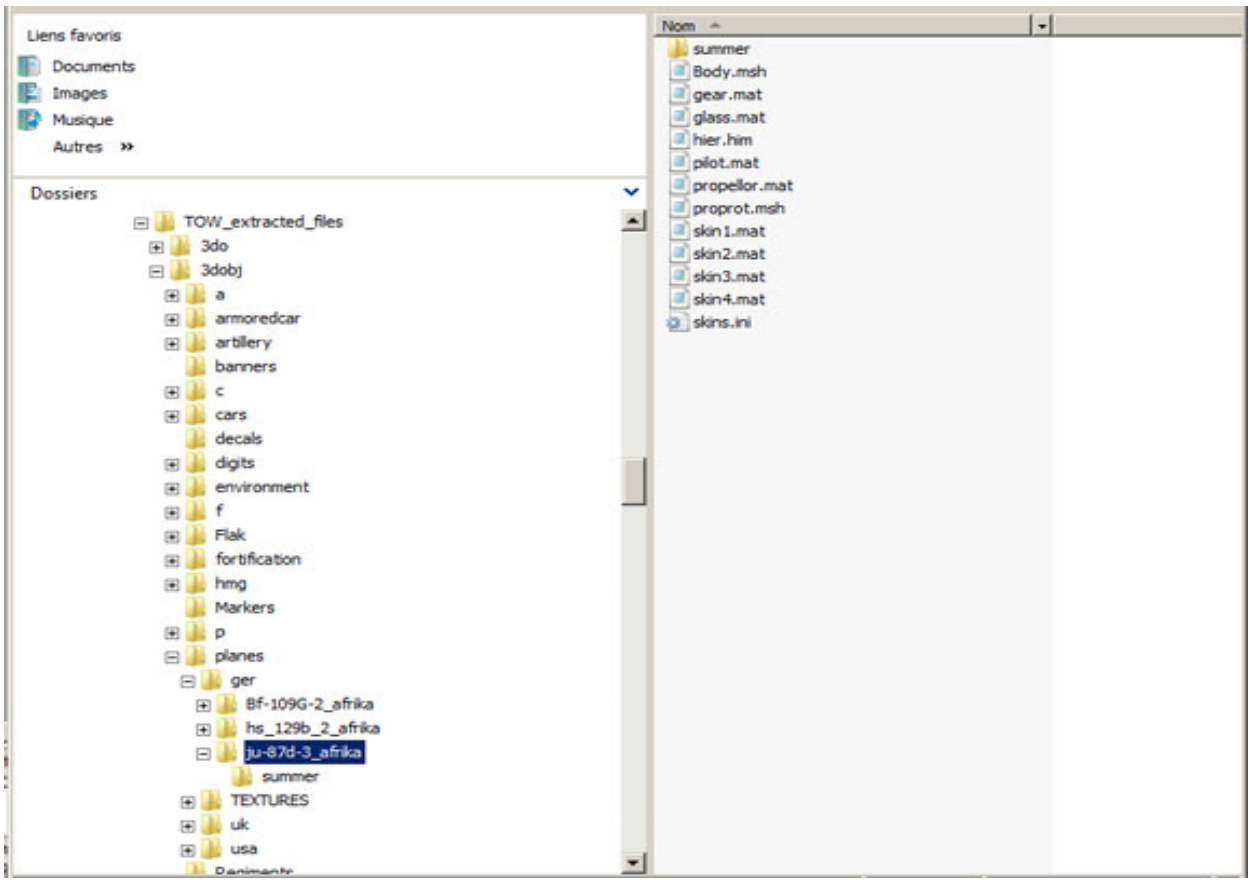
^ The recreated directories for a new typhoon plane inside the ToW main folder.

For creating a functional new unit in game, we'll need to work with three kind of files: *.DDS image files for the textures of our unit, text files with various extensions like *.him, *.mat, *.ini (various configuration files that can all be edited with notepad), and *.msh files (3D models in ToW format). These *.msh files are almost the same files than in 1C's game IL-2/Forgotten battles.

So to create a new unit, we'll need to create the same files that are used by an other similar unit already in the game, and set them up in the game structure in the same way.

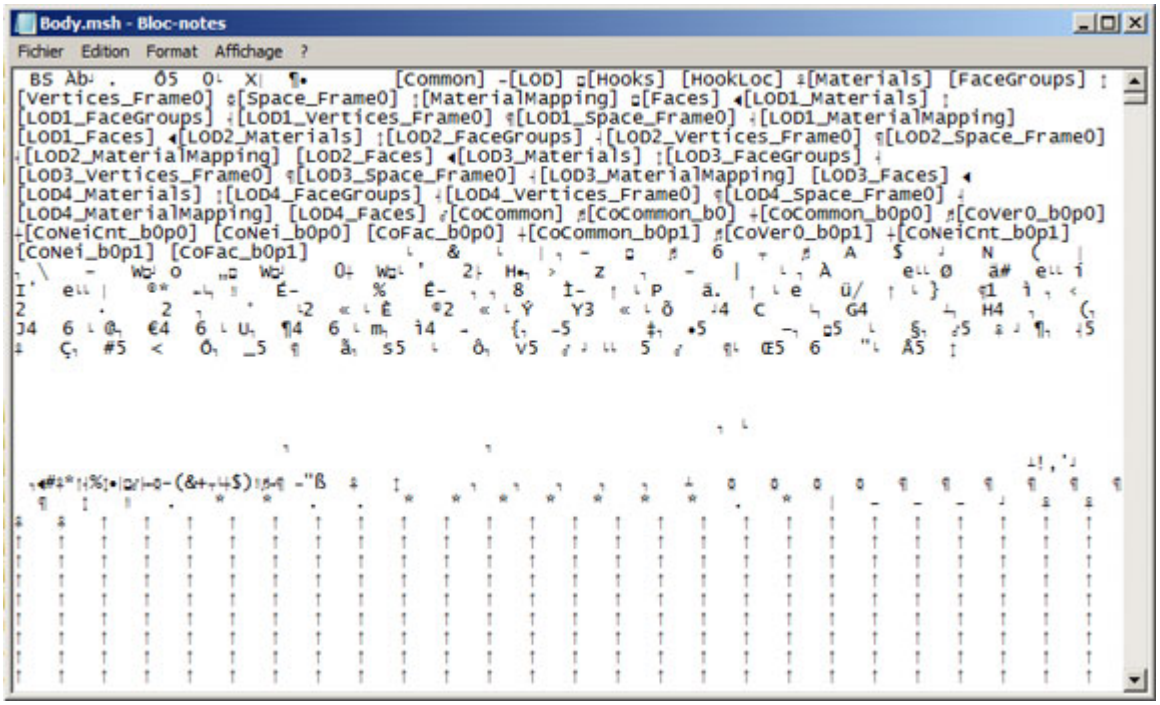
Looking at the files constituting a 3D model from the game:

First, we'll have a look at what files constitute the 3D model of an unit in game. As an example, we'll open the folder of a plane in the game: the Stuka (planes are the most simple objects to understand in the game, in my opinion).
We'll find the files in the folder: TOW_extracted_files/3dobj/planes/ger/ju-87d-3_afrika



^ The stuka 3D model files

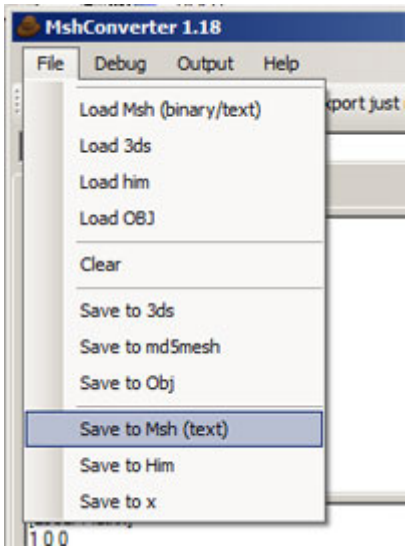
Have a look at the files existing in the Stuka folder: We have two *.msh files: Body.msh and proprot.msh (these are the 3D models of the plane and of the propeller). You could open the *.msh files in Notepad, but the file is in a binary format, which means that most of the content will not be readable:



^ The binary Body mesh file opened in Notepad.

If you want to have access to the content of the msh files in text format, you have to load this msh

in MshConverter and save it as a msh/text file:



Now you'll be able to have access to all the information contained inside the 3d mesh in text format. More on this later.

We have several text files that can be opened in notepad: the files with the extension *.mat. These are the files for setting up the materials of our unit and determining which texture a material will use.

The skins.ini file: this is a text file you can open in notepad. It contains the path of the textures that the 3D model is using. It is also used to set the regimental signs for the model. You can set the model up here to have several different texture sets (theatre or period dependant, and/or season dependent – for example one or more Summer and Winter textures sets)

There is also a folder Summer: this folder contains several *.mat files, skin1.mat to skin4.mat (setup of the materials for the plane) and corresponding textures files with a *.tga extension, skin1.tga to skin4.tga (textures that will be displayed on the plane's models). Each of theses mat files and textures correspond to one LOD (level of detail) of the airplane model. Skin1.mat and skin1.tga are for the first (most detailled) LOD, skin4.mat and skin4.tga are for the less detailled LOD.
If you try to open theses *.tga files in photoshop, you'll get an error message, because they're not really Targa images, but *.DDS images (compressed textures files of the type [DXT3 ARGB 8 bpp | explicit alpha] in photoshop DDS plug-in). If you change the extension from *.tga to *.dds, you should be able to open the files in photoshop if you have installed the Nvidia photoshop plug-in to work with *.DDS pictures.
If you're not using photoshop, you can still convert the DDS files with IMFViewer to a readable true TGA format. ToW should accept formats other than *.DDS (like it is the case with the maps textures files like Maintex.tga and Farmaintex.tga), but I've not experimented with it, and I've stuck to the formats used in the game. Not all objects will have DDS textures with a TGA extension; most of them will have kept their *.DDS extension. ToW engine should'nt care of the extension you're using, as long as the file is correctly referenced in the *.mat files.

The last file to be found in the Stuka folder is a file called hier.him. You can open it with Notepad This file is used by ToW engine to assemble the 3D models of the plane. The plane is a simple object, and is made of only two *.msh: one for the « body » of the plane, and one for the propeller. The hier.him file contains all the hierarchical data to assemble the various parts of the 3D models. It contains also information about the collision objects that the model will use:

[_ROOT_]
VisibilitySphere 9.40860
CollisionObject sphere 9.40860 0.0 0.0 0.0
BoundBox -6.557688 -7.521715 -2.434510 5.548208 7.514601 2.457101
[Body]
Mesh Body
Parent _ROOT_
Attaching 1 0 0 0 1 0 0 0 1 -0.50474 0.022017 -0.047089
CollisionObject x_Hull
CollisionObject x_Wing
[proprot]
Mesh proprot
Parent Body
Attaching 1 0 0 0 1 0 0 0 1 5.55744 -0.0255733 0.115823

[_ROOT_] This group is the origin of the 3D model (coordinates 0,0,0)

[Body]: this group is the fuselage and wings of the Stuka and correspond to the placement of the Body.msh part.

[proprot]: this is the group of the propeller of the Stuka, and is related to the placement of the proprot.msh part.

Under each of the groups Body and proprot, there is a line starting with the word « Attaching » followed by numbers.
This is the transformation matrix for the part.
[sx 0 0 0]
[0 sy 0 0]
[0 0 sz 0]
[x y z 1]
the « s »here stands for Scale (you can change the scale of your model here, but I believe it is better not to play with it
The first three groups of three numbers (100 010 001) are for the scaling and rotation of the part. The last three numbers are the translation values for the part . 0,0,0 would mean that the part would be centered on the origin. Here for the proprot group, the values 5.55744 -0.0255733 0.115823 would mean that the propeller will be attached at the correct position on the propeller hub of the plane.

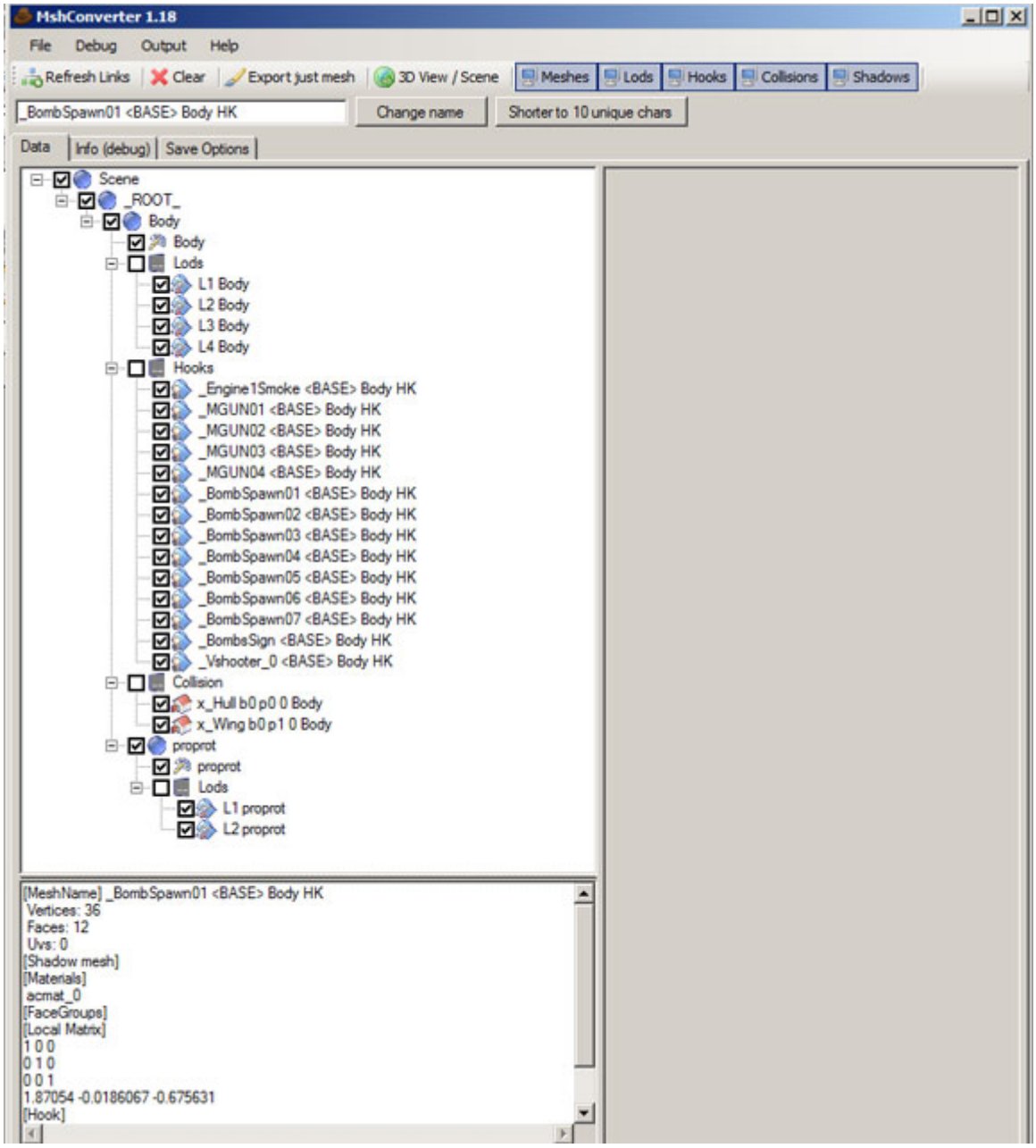
In the [Body] group, we can find the two collision objects being used: x_Hull and x_Wing.

Loading a 3D model from the game in Mshconverter for reference:

The best way to understand how ToW models are organized is to open one of the game model with Dr. Jones MshConverter. As an example, we'll load the model of a plane in the game: the Stuka. We'll find the files in the folder: TOW_extracted_files/3dobj/planes/ger/ju-87d-3_afrika

Launch MshConverter and open the menu file-> load him. We could open only one of the two

*.msh constituting the plane, but we'll load the hierachy file instead, to see how the whole model is organised:
So select File-> Load him and open the hier.him file located inside the Stuka folder.



^ The whole stuka hierarchy opened in MshConverter

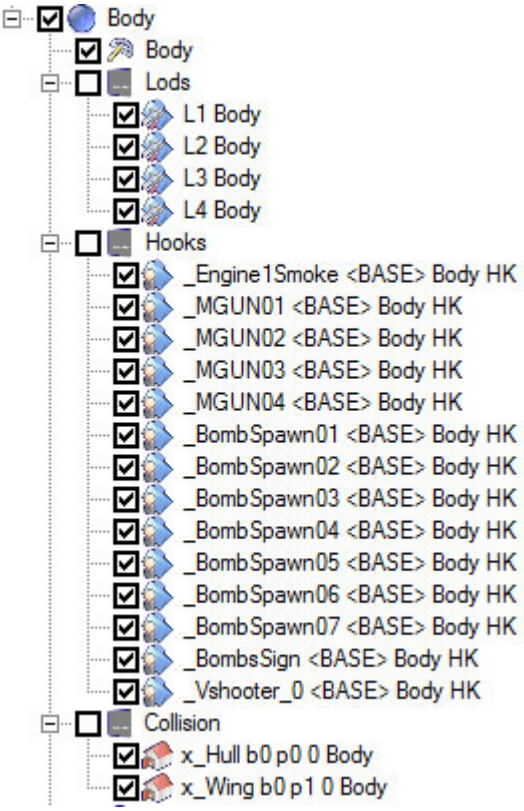
Now, this looks like a bit more complicated than before, because a lot more objects are appearing that we could'nt see in the hier.him file. The hier.him file is only for assembling together *.msh files. Each *.msh file contains several different objects.

Groups or Bones:

Let's have a look at the top of the hierarchy:

First, we have the _ROOT_ group with a blue sphere in front (this is the group that we found in the hier.him file). This is the origin of the model centered at coordinates 0,0,0

Just under the group _ROOT_ we find the group Body. Groups have a blue sphere in front of them, while other entity have a different icon, according to their type.



Main mesh object:

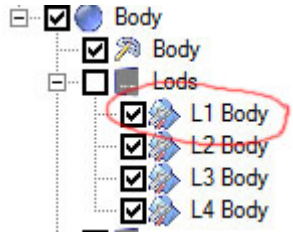
Just under the group Body, we have an other entity with the icon corresponding to a « main mesh ».

^ The main mesh object

A main mesh object is a LOD0 object; that is , the most detailled model, and so the one that is seen when we are the closest from the unit.

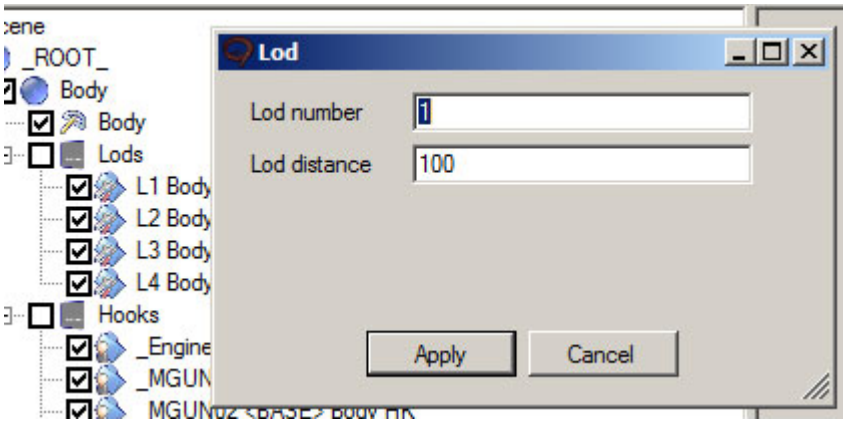
Lods objects:

Each object constituting an unit must have not only a main mesh (lod 0), but also several other lods with a lesser resolution (less polygons). The further away the model, the less polygons it should have. The Theatre of War engine will decide which one of the LOD to display, depending on the distance of the unit from the camera, and which complexity setting have been choosen by the user. There is no need to have a fully detailled model being displayed on the screen when an unit is so far away from the camera, that we can barely see it. So to save calculations, a model with less and less polys is used the further away from the camera it is. Our plane have four other lods that we can find under the « Lods » entry in the hierarchy:



^ The Lod1 for the Body mesh.

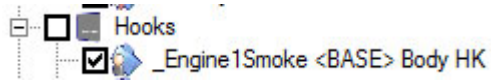
They are named from L1 to L4. If you left/doubleclic on one of the lod, a popup will appear, where you can see and eventually modify the LOD number, and LOD distance of appearance:



Here, the most detailled Lod (main mesh/Lod0) will be displayed from 0 to 100 meters. The main mesh/Lod0 will dissappear, and the Lod1 will be displayed instead as soon as the camera is 100 meter away from the unit. The prefix « L1 » to« L4 » allows MshConverter to determine that the object is a LOD, and to know what is the priority of the Lods and so, in which order they'll be displayed.

Hooks objects:

Further down the list, under the group Body and just after the Lods, we find the Hook objects:



^ The engine smoke hook object.

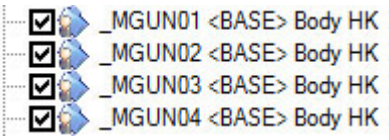
Hook objects are just small polygons (the shape of the hook object doesnt really matter; it can be a

simple triangle, or a cube, or anything else that is convenient – it can be a pointed arrow if the orientation of the hook does matter, for ease of reference. This is often the case for a gun or mg hook to indicate the orientation of the barrel. But it doesn't have to; a simple cube is enough).

Hooks are use to determine where an effect will be « spawned » by the game engine on the model. The center of the hook object is used to determine the placement of such an effect on the model.

Here, the _Engine1Smoke <BASE> Body HK object purpose is to indicate where the smoke and flames will originate when the plane is shot down and is trailing smoke. Note that the naming convention of the hook is very important for the game engine to recognize what the hook is made for. Note that the names are case sensitive (unless I'm mistaken). The orientation of the smoke hook should be with the x axis oriented toward the rear of the plane.

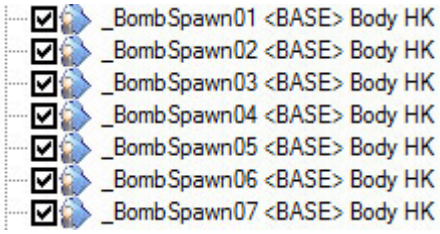
Next, we have the hooks for the machine guns, named __MGUN01 to _MGUN04 :



^ The machine guns hooks objects

_MGUN01 <BASE> Body HK is for the right wing MG17, _MGUN02 <BASE> Body HK is for the left wing MG17, while _MGUN03 <BASE> Body HK and _MGUN04 <BASE> Body HK are for the twin MG81Z of the rear gunner. Note that these hook objects have to be oriented so that the x axis is pointing toward the direction of fire, and the center of the object should be at the end exit of the gun barrel.
Theses hooks are used to position the effects of the flames and smoke originating from the barrel when the aircraft is shooting.

Next we have the _BombSpawnXX hooks:



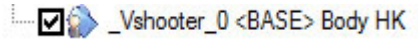
^ The bombs hook objects

There is one hook for each bomb carried. These hooks are used to position the bombs on bomb carying support aircrafts. The bombs will be displayed under the wing at the center of the hook, and will remain there until dropped. Note that the orientation is also important here.

Next we have a hook whose function I don't know yet: _BombsSign <BASE> Body HK
It is located at almost the same position as the heaviest bomb, under the belly of the aircraft.

^ The BombSign hook object

And finally, we have the _Vshooter_0 <BASE> Body HK hook object:



^ The Vshooter hook object.

This hook correspond to the vision of the pilot. It is the equivalent of the hooks used for the sighting devices and observation devices on the vehicles and tank units.

MshConverter is able to recognize objects as hooks because they have the suffix « HK » at the end of the name.

Collision objects:

Next in the list are the collision objects:

^ The collision objects

Collision objects are identified as such by MshConverter because they have a prefix starting with an « x ». This is causing some trouble for many objects types, because a lot of static objects have a main mesh called something like x0y0_dam0, and MshConverter will confuse theses main meshes with collision objects because of the « x » at the start of the name. This is not really a problem, as by right clicking with the mouse on an object, you can specify what is the type of the object, and correct this problem when MshConverter automatically detect a main mesh whose name is starting with an x and mistake it for a collision object. So if one of your objects is not detected as being of the correct type, simply right-click the object, and set the appropriate type for it in the pop-up menu. Then click the refresh link button located on the upper part of MshConverter window, just under the « File » menu. MshConverter will then reorganize the hierarchy, so that the entity will now be displayed in the appropriate group.

A collision object is a very simple 3D object used to allow the game engine to detect collisions with bullets and shells, and also to detect if something is visible or hidden behind an other object. If a soldier is located fully behind the collision objects of a house for example, he will not be visible, and bullets won't be able to arm him (this depends also of the armor value of the object, as some shells are able to get through the collision objects, depending of the type, armor values, angle...)

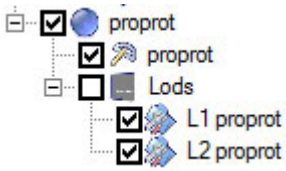
A collision object must be a convex object for the game engine to detect the collision correctly. If not, some strange bugs may appear. The object must be fully closed, and have as low a poly count as possible (10 to 25 polys at the most). For a complex object, one will need several different collision objects to represent all the collisions. The game engine will combine them and use them together to detect collisions. Collision objects must intersect each other for the collisions to work correctly (unless they are not touching eachother, of course). For our Stuka, we need a collision object for the fuselage, and one collision object for the wings. You don't need a lot of collision objects, keep the number as low as possible to ease calculations.
There was a limitation in ToW1: one « Bone » (I called this a group, because for me a bone usually is a 3D entity used to skin meshes in 3D applications) must have no more than 7 collision objects. If you need more collision objects, you have to create one or several other Bones with the names « dummy01 » to « dummyXX » for exemple (if you really have a lot of collisions). I'm not sure this is still needed in ToW 2, as the houses objects seems to have more than 7 collision objects per bone.

I'll not dwell any longer o the collision objects, as more information can be found in the 1C document « Unit modeling guide for Theater of War V.1.0 eng » mentionned at the beginning of this tutorial.

All collision objects will have a name ending with the same kind of suffixes: b0 p0, or b0 p1
Theses are incremental. The first collision object will alway end with b0 p0, while the second will be ending with b0 p1, the third with b0 p2 … and so on.

Second Bone:

The next group that we can find in the hierarchy of the Stuka is the « proprot » group:

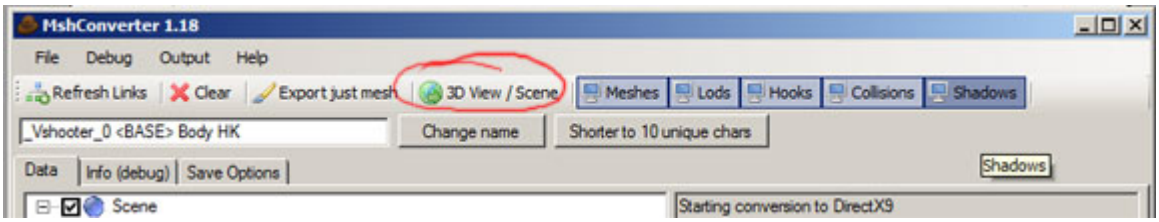


^ The proprot bone

The proprot bone is used for the spinning propeller mesh.
By now, you should be able to figure out easily what theses objects are:
we have first the proprot main mesh, and next the two lods for this main mesh.
Note that there are only two lods for the circle of the spinning propeller, as from longer distance, it would be unnecessary to display the rotating propeller.

Using MshConverter 3D viewer:

If you look at the top of MshConverter window, you'll see a button called 3D view/Scene. Clicking this button will open MshConverter 3D viewer, allowing you to look at the 3D objects located in the scene:



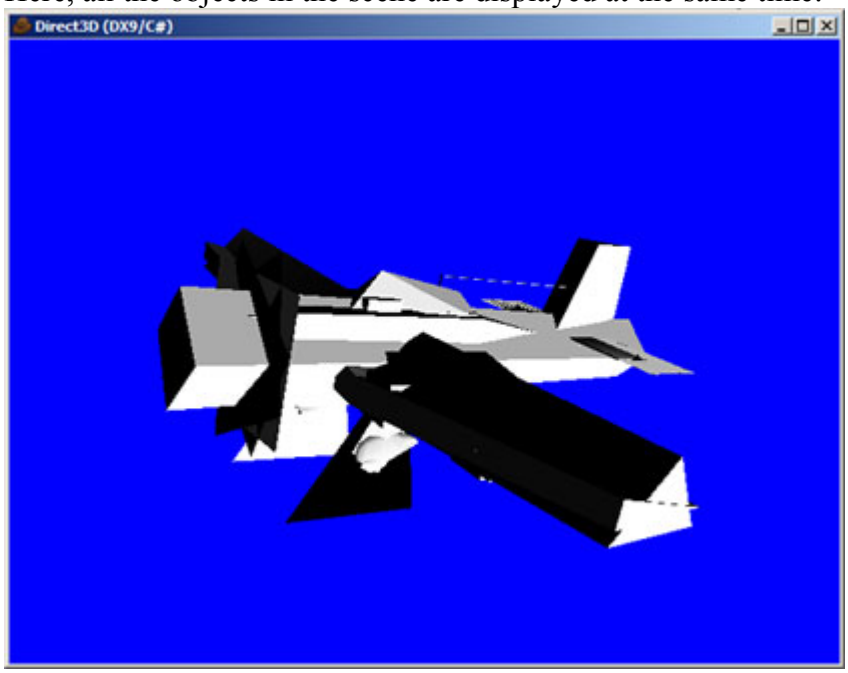
^ The 3D view/scene button circled in red.

You can choose which part you want to display in the viewer by checking or unchecking the box in front of each entity in the hierarchy. If you uncheck the box in front of a bone, the whole objects inside the bone will be hidden.

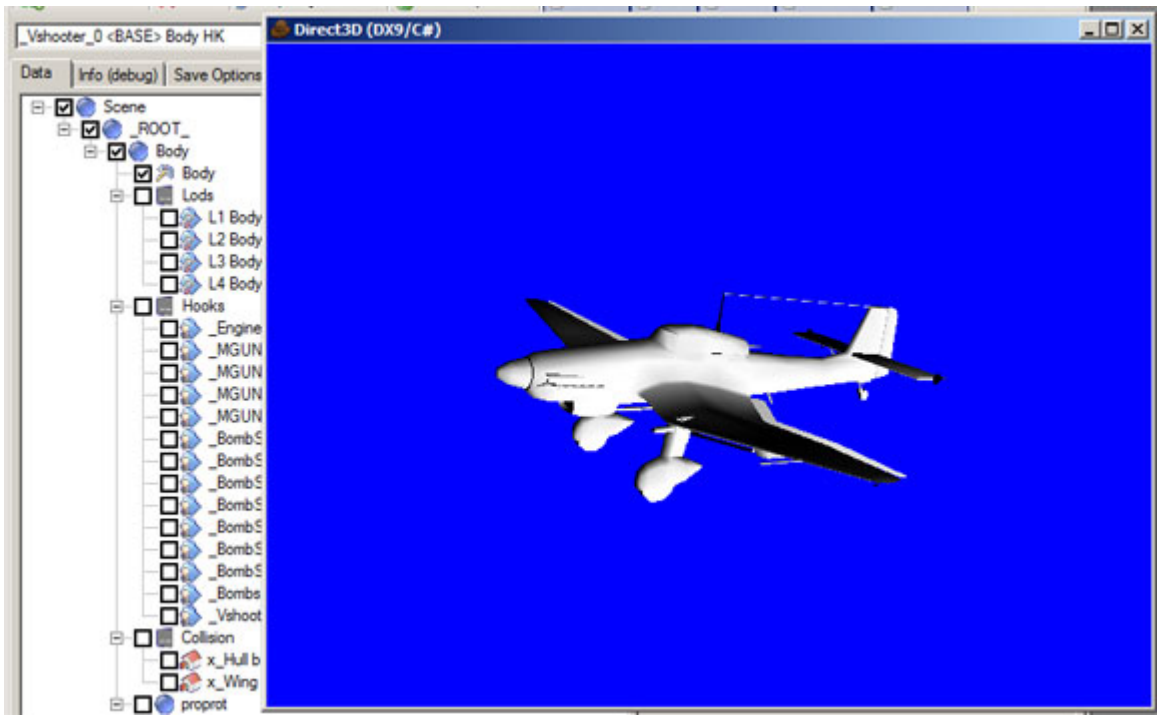
You can change the camera angle in the viewer by keeping pressed the middle mouse button/wheel

and moving the mouse at the same time.
You can zoom in and out by rotating the mouse wheel.

Here, all the objects in the scene are displayed at the same time:

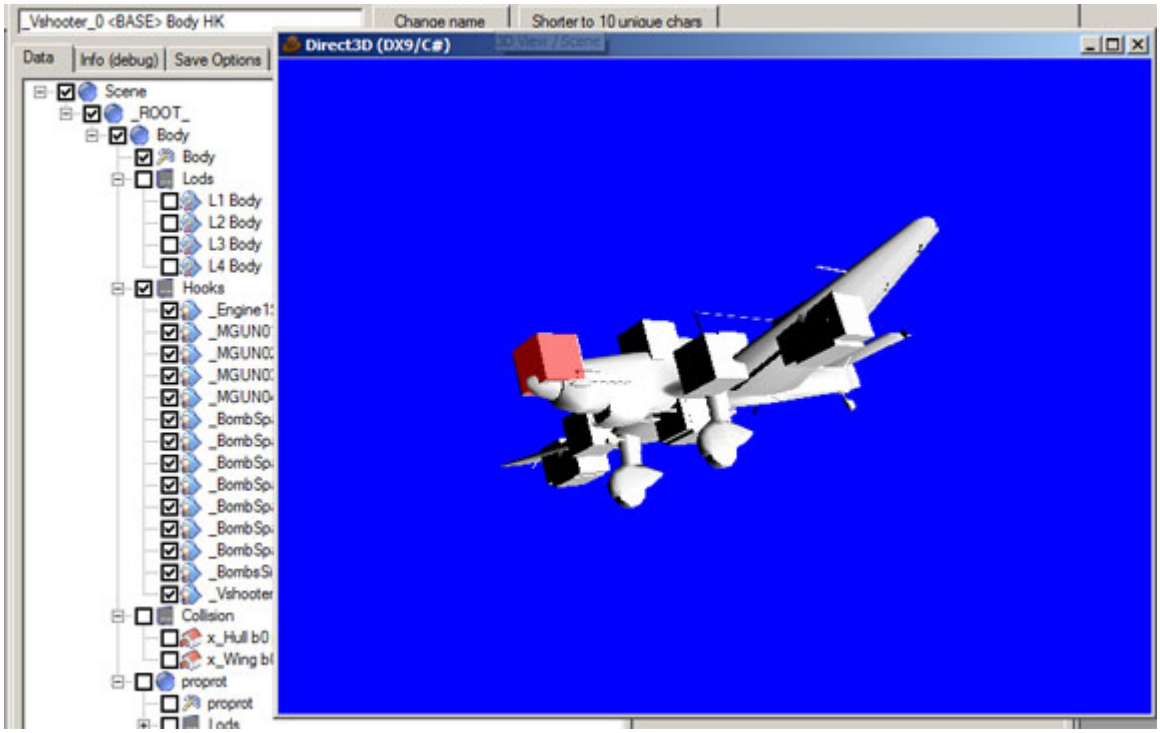


Here, only the main mesh is displayed:



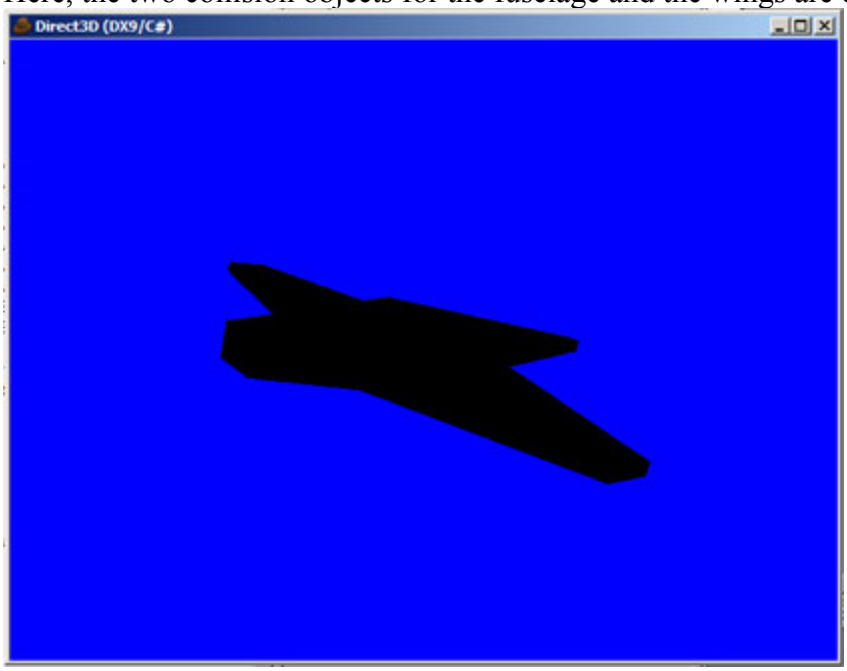
See the box in front of the Body main mesh is checked, while all the other boxes are left unchecked.

The hook objects displayed at the same time as the main mesh:



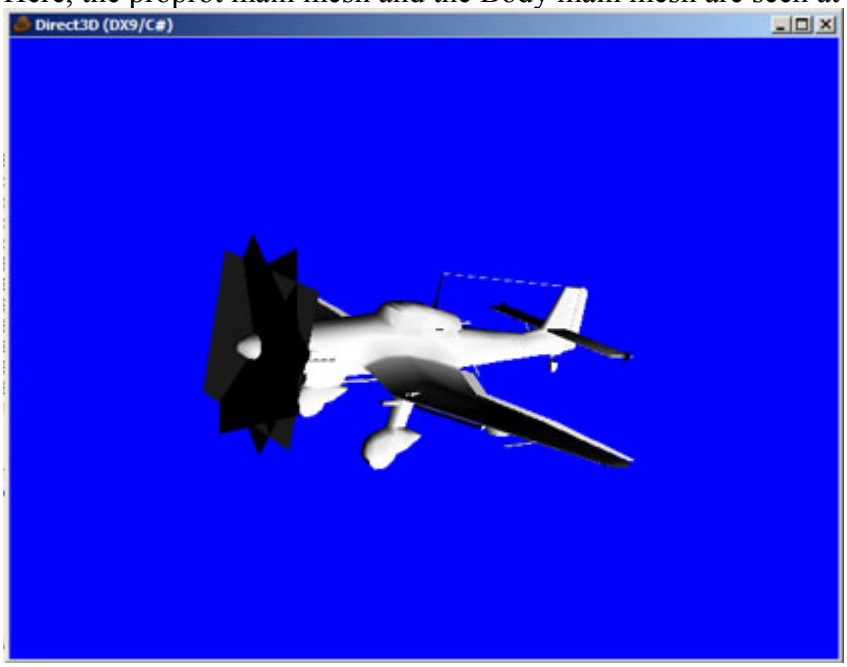
I have colored the _Engine1Smoke hook in red. Hooks are represented as cubes by MshConverter.

Here, the two collision objects for the fuselage and the wings are displayed:

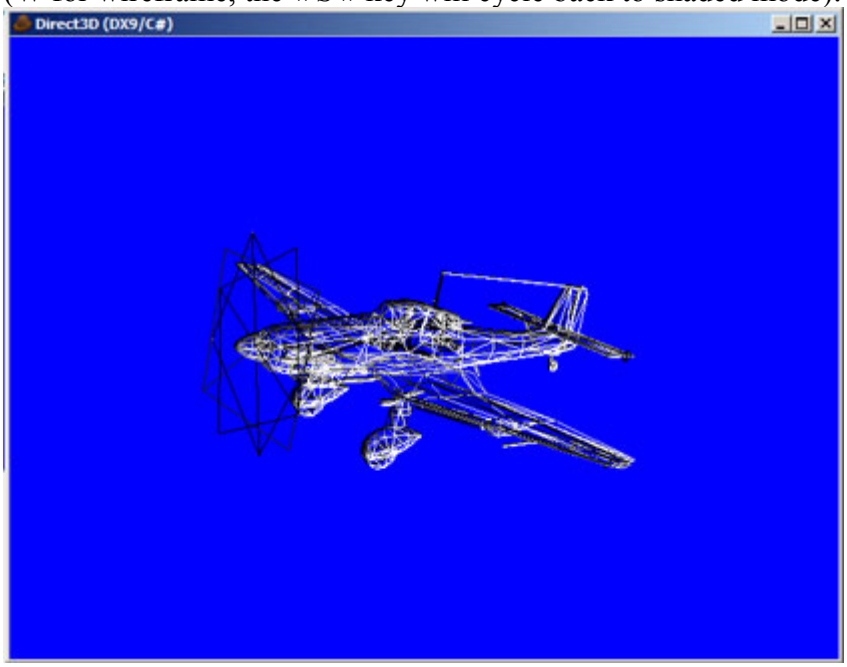


The viewer allows you to see all the 3D objects in the scene, compare their position relative to each other, and see the location of the hook objects easily. Most of the time, you'll have to switch a lot to the 3D viewer, when working with objects with more « esoteric » names like x0y0_dam0, x0y0_dam1, x0y1_dam0 and so on... , if you want to be able to keep track of which name correspond to which object.

Here, the proprot main mesh and the Body main mesh are seen at the same time:

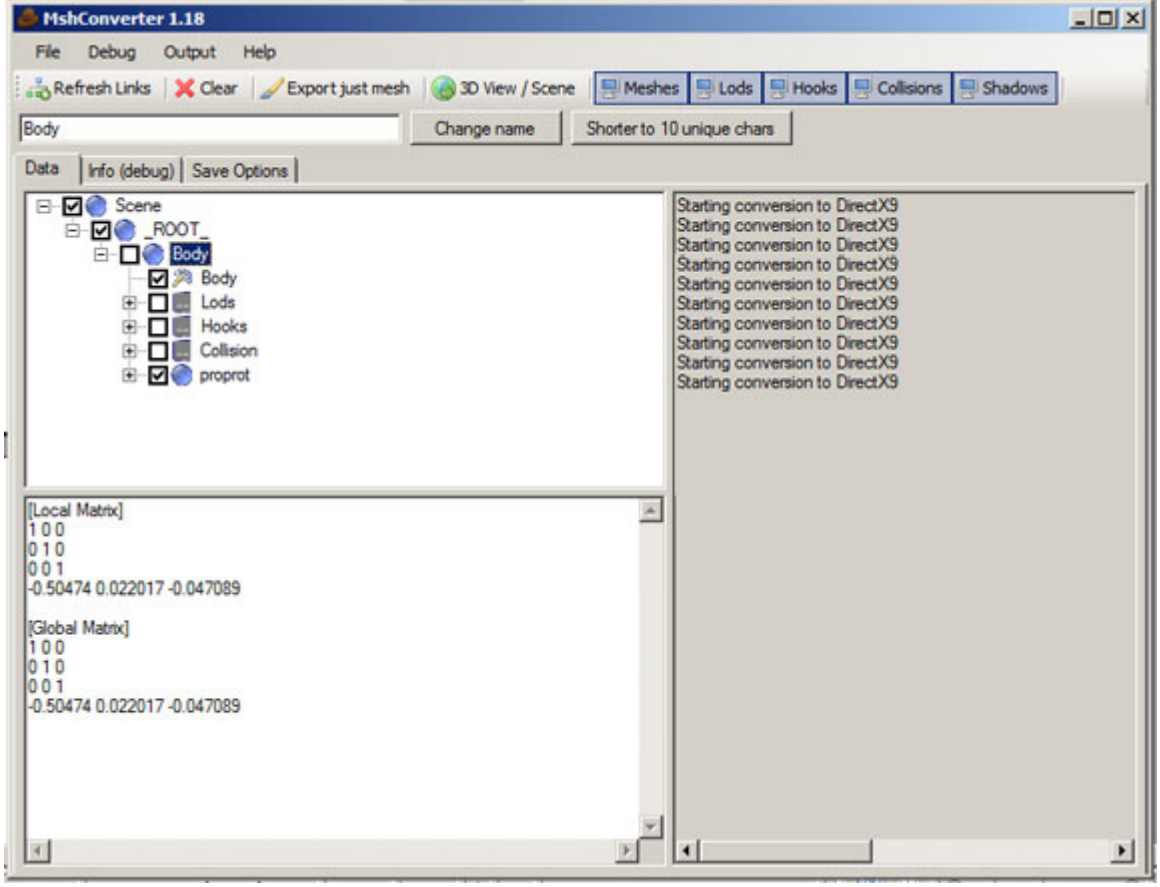


You can also enable the wireframe mode in the viewer by hitting the « W » key on your keyboard (W for wireframe, the « S » key will cycle back to shaded mode):



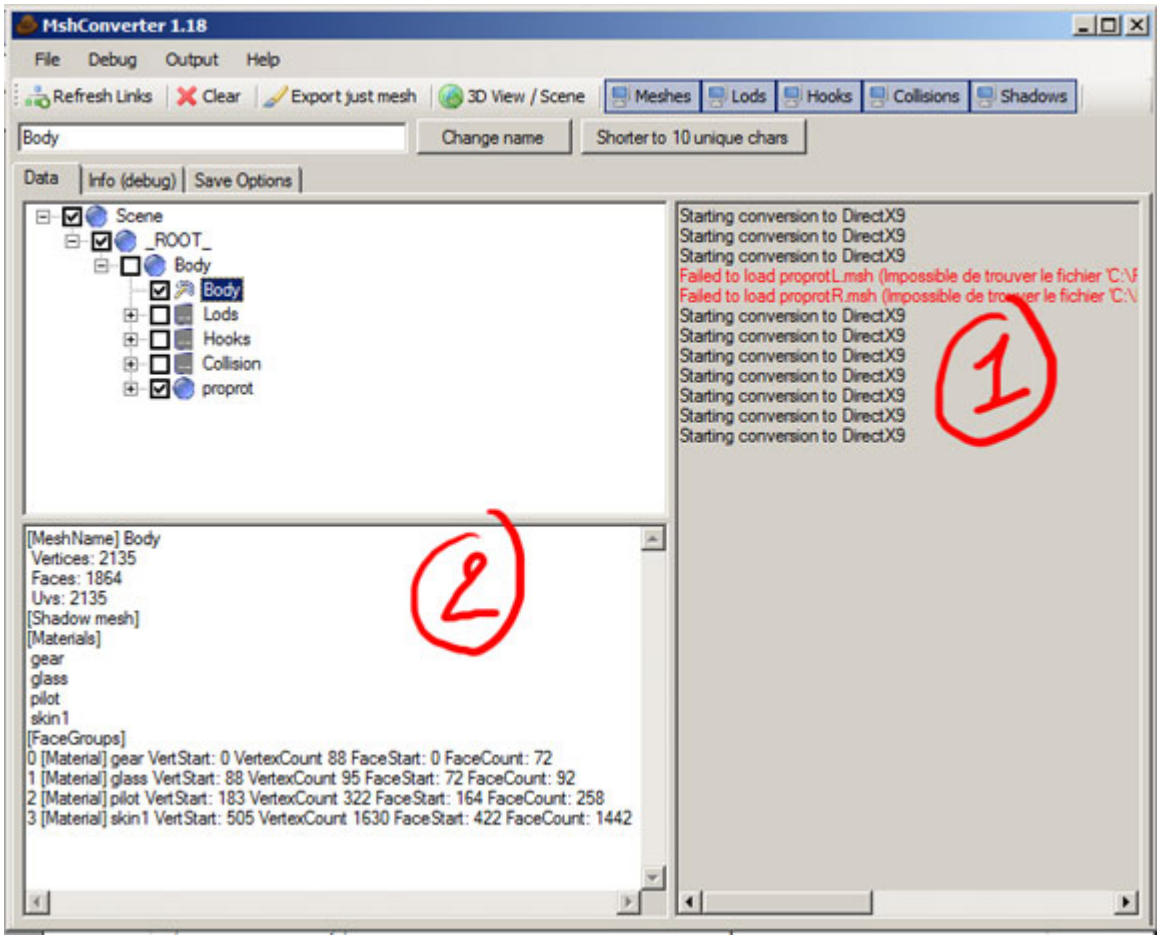
MshConverter will be your main tool to try to understand how the 3D objects in the game are organized, what kind of hooks they need to be functional, how many and how the collision objects are organized. MshConverter also allows you to get information on the various objects in the scene:

The objects info window:



Here the bone Body is selected: in the lower part of MshConverter window, the transformation matrix for this bone is shown. This is the same transformation matrix as seen in the hier.him file.

If you select a mesh, you have access to the information for that particular mesh:



In this dialog window, you'll see written in red any error message generated while loading the meshes (Number 1 in the above picture).

Number 2 is the information displayed about the Body main mesh:

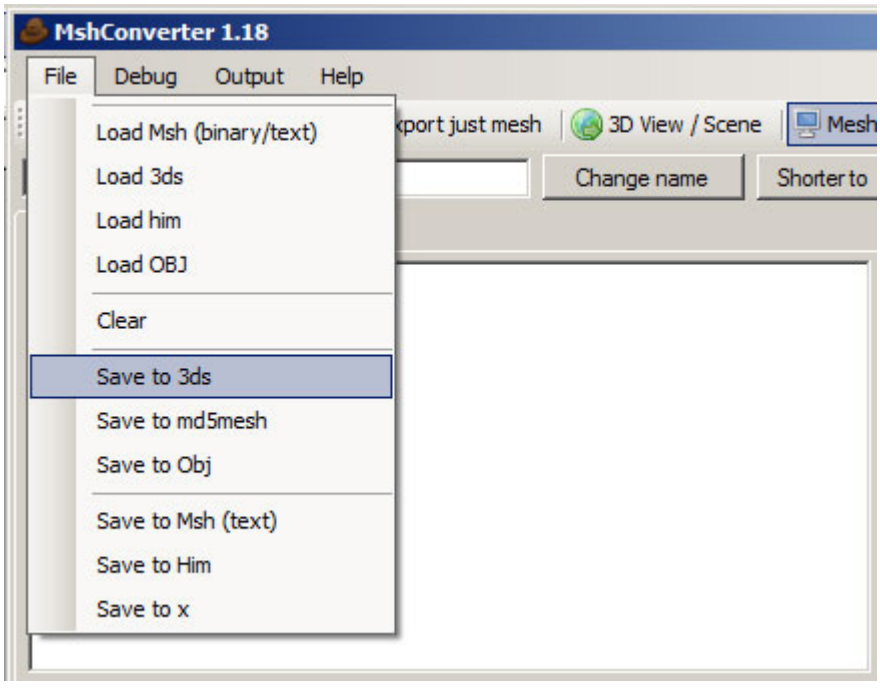
The name of the mesh
number of vertices, faces, and uvs
If there is a shadow mesh
a list of the materials used in this particular mesh
a list of the FaceGroups (all the faces using the same material ID)

There, the mesh is using four materials/textures: gear for the wheels, glass for the cockpit, pilot for the pilot and gunner, and skin1 for the texture of the fuselage and wings.

Information on the number of polygons and material allows you to get a good idea of the number of polys you should use when designing your own objects. Try to keep as close as possible to the numbers found in the default ToW objects.

Exporting a model to 3DSmax with MshConverter:

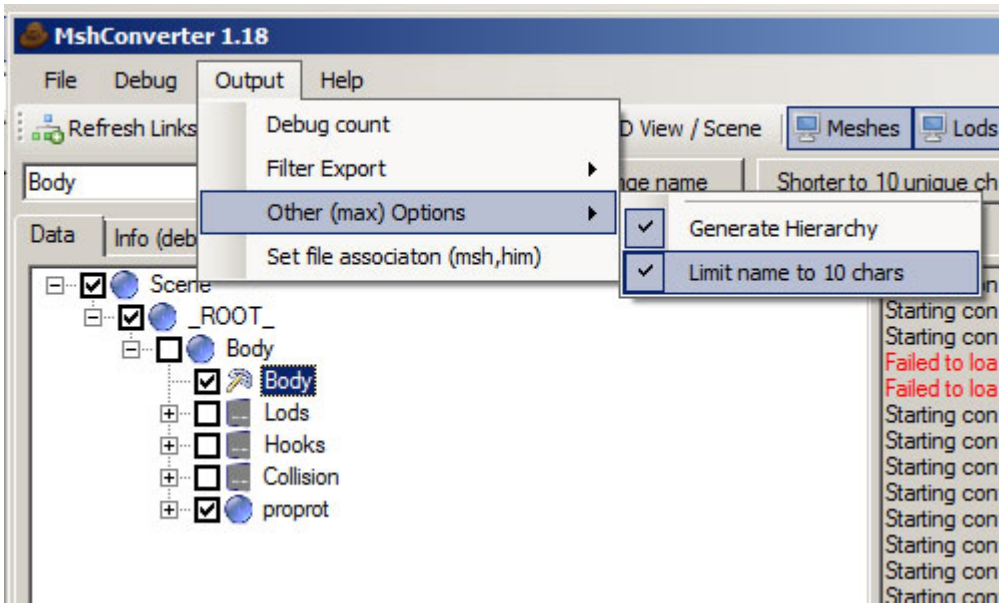
You can export the Stuka hierarchy to 3DSmax by way of saving to a *.3DS file.



But first, we have to consider a limitation of the 3DS format: objects names can be no longer than 10 characters. As you can see in the hierarchy displayed in MshConverter, a lot of the objects have names that are a lot longer than that. Therefore, we'll have to shorten the names so they are no more than 10 characters long, or else 3DSmax will give us an error message and will not open the files.

Renaming the objects for export:

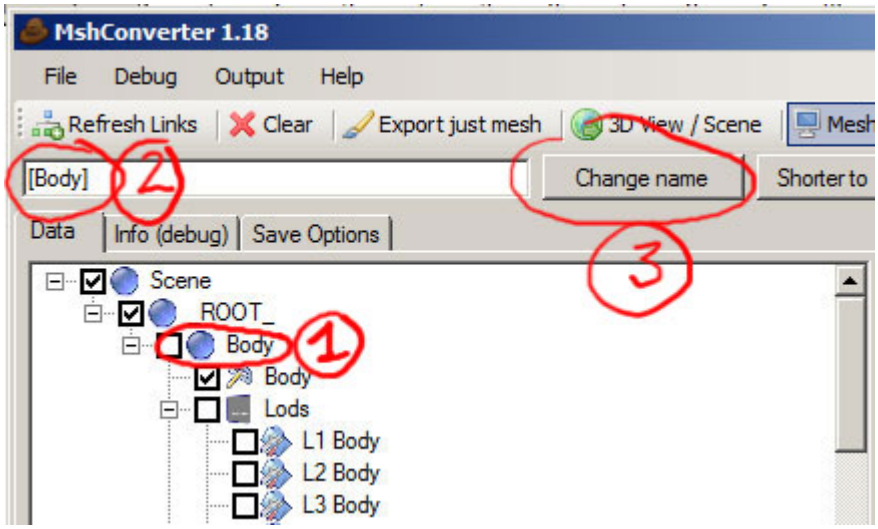
Fortunately, there is an option in MshConverter to shorten the names automatically:



You can find this option in the menu Output on top of the MshConverter window.
The option « generate hierarchy » is checked by default, and we need it, because we have loaded the hierarchical file hier.him, and not just a mesh. We need to generate this hierarchy if we want the 3DS file to be organised accordingly.

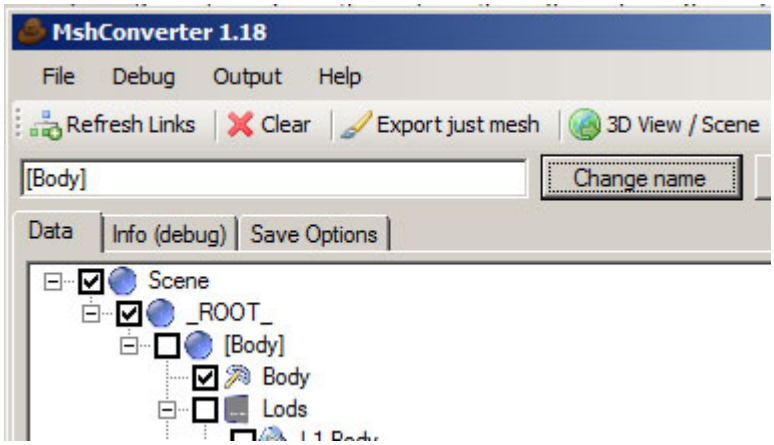
If like me, most of the time you're not pleased with the sorther names that MshConverter is giving the objects, you can rename them manually to something more understandable (to keep track of which object it is, when you have a lot of objects with the 10 first characters being the same.

To do that, simply click on the object whose name you want to change. Here, I'm going the rename the Body bone to [Body], in order not to have two objects with the same name, and to be able to easily differenciate the Bone from the Main mesh (not that you'll really need it, a bone and a mesh being easily reconized)



On the above picture:

(1) click on the name of the object you want to change (here the bone Body)
(2) The name will appear here, and you only have to type the new name
(3) hit the Change name button, and the name will be modified in the hierarchy.



^The name of the Body bone have been changed to [Body]

Next, rename all the object that have a name longer than 10 characters. A space in a name counts as a character.

This means that we'll have to change the names of all the hook objects, as they are way over 10 letters long:

for example, you could rename them like this:

_Engine1Smoke <BASE> Body HK ----------> _Engin1 HK
_MGUN01 <BASE> Body HK ----------> _MGUN01 HK
_MGUN02 <BASE> Body HK ----------> _MGUN02 HK
_MGUN03 <BASE> Body HK ----------> _MGUN03 HK
_MGUN04 <BASE> Body HK ----------> _MGUN04 HK
_BombSpawn01 <BASE> Body HK ----------> _BombS1 HK

_BombSpawn02 <BASE> Body HK ----------> _BombS2 HK
_BombSpawn03 <BASE> Body HK ----------> _BombS3 HK
_BombSpawn04 <BASE> Body HK ----------> _BombS4 HK
_BombSpawn05 <BASE> Body HK ----------> _BombS5 HK
_BombSpawn06 <BASE> Body HK ----------> _BombS6 HK
_BombSpawn07 <BASE> Body HK ----------> _BombS7 HK
_BombsSign <BASE> Body HK ----------> _BombSi HK
_Vshooter_0 <BASE> Body HK ----------> _Vshoot HK

I tried as much as possible to give them new names easy to recognize. I also deliberately kept the suffix HK (separated by a space) at the end, so that when re-importing from max to MshConverter, MshConverter detect the hook objects correctly as belonging to the type Hook.

Next for the Collision objects. You should keep the x prefix in front of the names for the same reason.

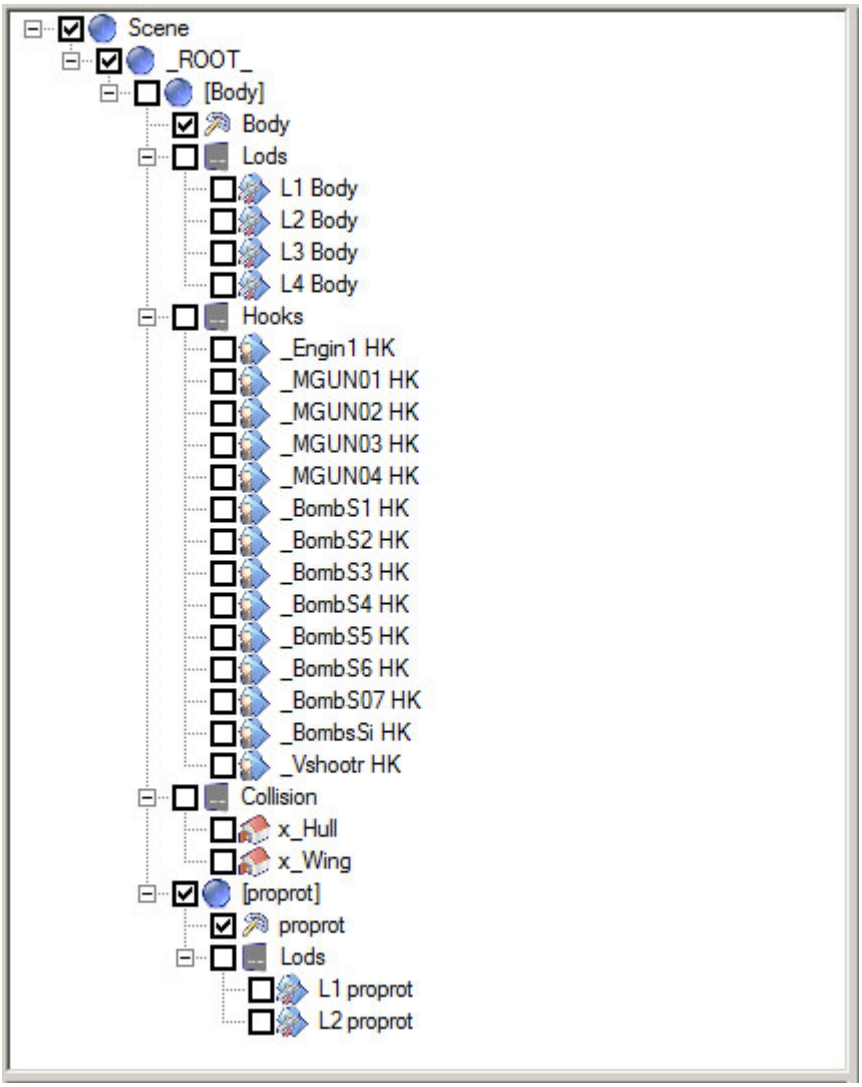
x_Hull b0 p0 0 Body ----------> x_Hull
x_Wing b0 p1 0 Body ----------> x_Wing

I'll rename also the proprot bone, so as not to have twice the same name in the hierarchy:

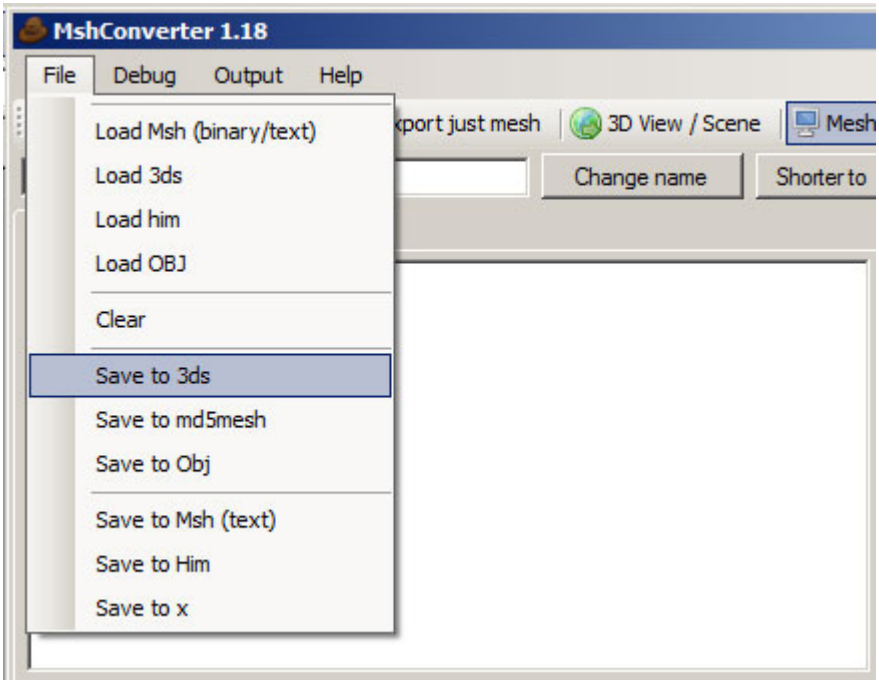
proprot ----------> [proprot]

Well, in fact you don't really have to do it, as MshConverter will modify the names automatically if there is twice the same name.

Here is a picture of the hierarchy in MshConverter after having shortened the names, ready to export as a 3DS file:



All you have to do now is exporting the model to 3DS format. Notice that you'll need to unhide all the parts, or only the visible ones will be exported.



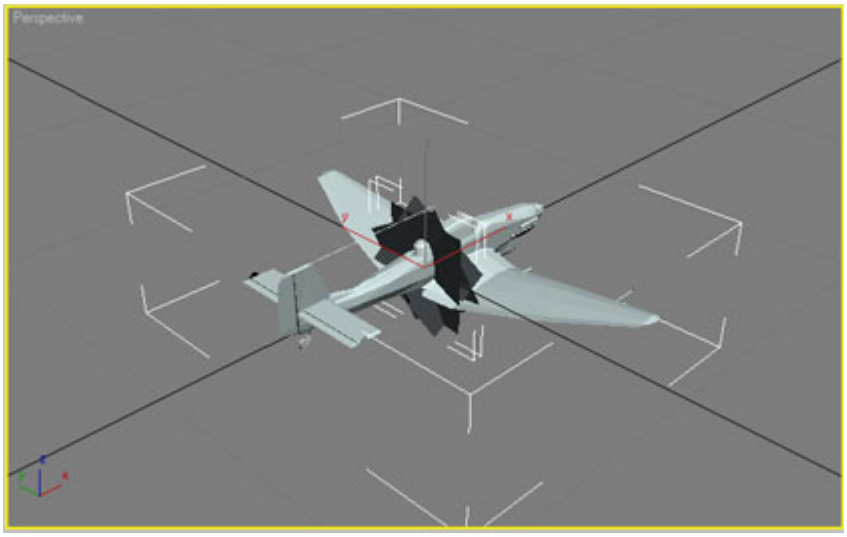
Save this 3DS file in a working folder you've created on your hard drive. The file will be given a default name of model.3DS, but you can rename it to whatever you want, so that you know which file it is.

You should now be able to open the file in 3DSmax.

Your model opened in 3DSmax:

Here I'll not explain how to use 3DSmax, so I assume you have at least basic knowledge of how the application is working. If not, browsing through Max basic tutorials should be enough to get you started.

When importing your 3DS file, you should have set 3DSmax to use meters as units setup, and you should keep the « convert units » option in the import 3DS dialog box unchecked.

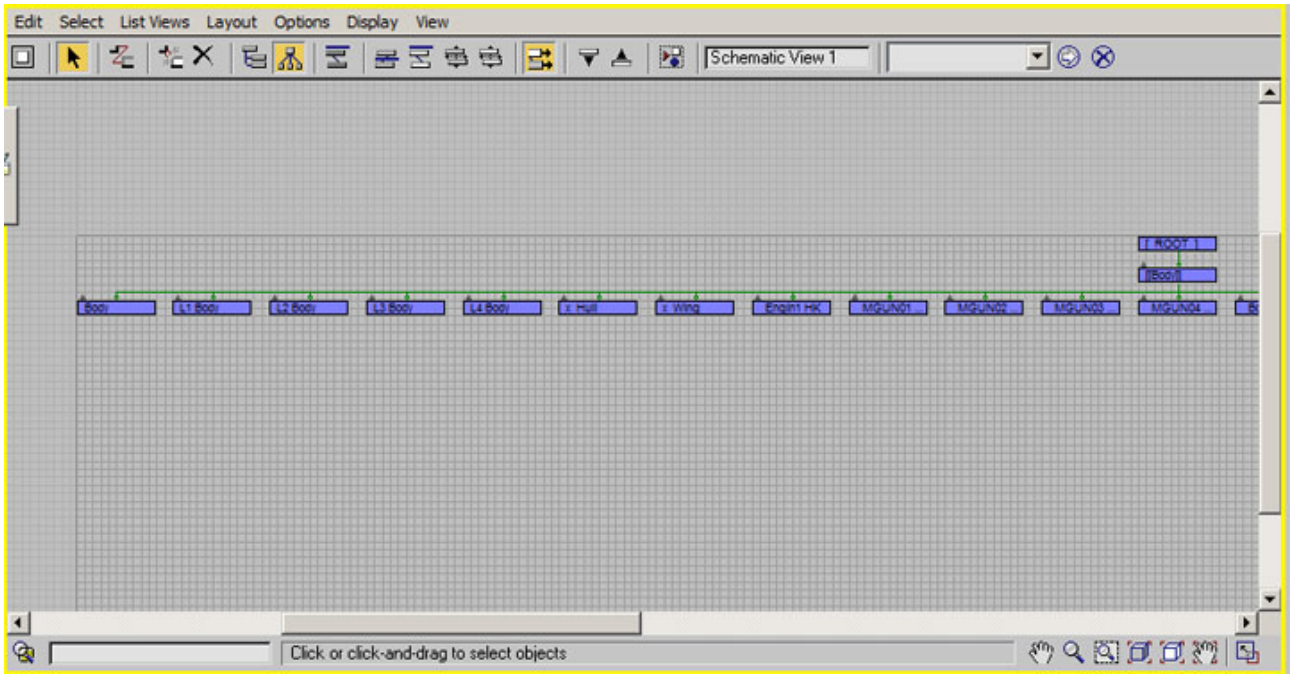


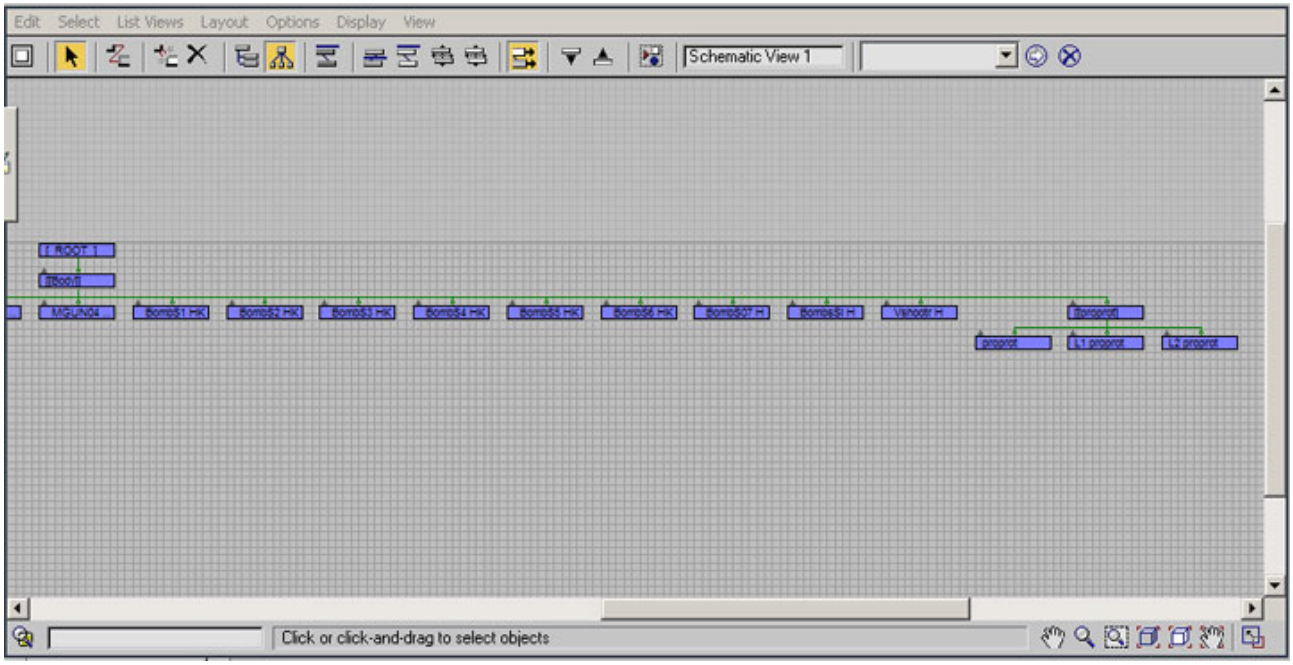
^ The main mesh Body and main mesh proprot in Max.

As you can see, the proprot bone is located at coordinates 0,0,0. It is the transformation matrix in the hier.him file that position the propeller on the propeller hub of the Stuka.

You could move the vertices of the proprot main mesh and the two proprot Lods to the correct position if you wanted, thus having no need to have an offset in the transformation matrix of the hier.him. But for this to work, the axis of the hub should be positioned at y=0 and z=0. This is not the case for the Stuka, as the propeller hub is a bit higher in the z axis. So you need to keep this offset in the transformation matrix if you want the propeller rotation to be centered on the propeller hub.

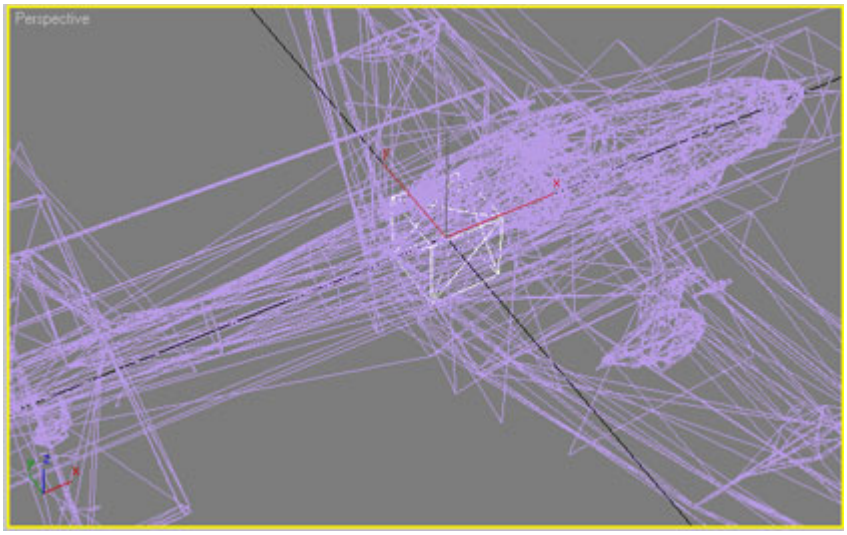
Here is the hierarchy of the Stuka after importation in 3DSmax:





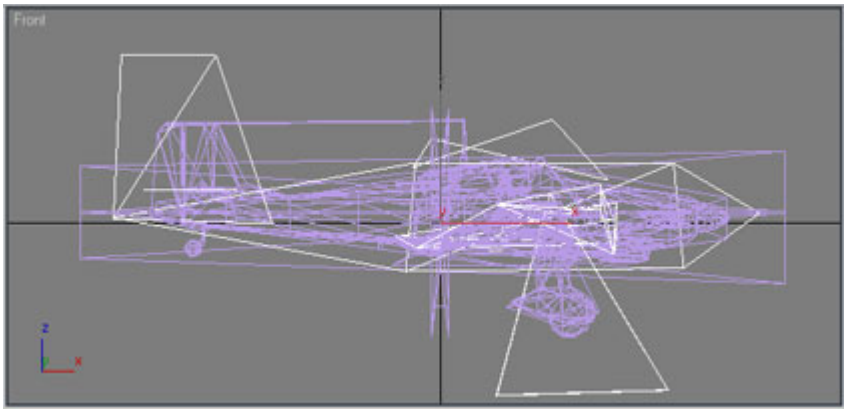
You can see the same organization of the objects in Max as the organization in the hier.him file and in MshConverter.
The _ROOT_ is the parent of the Body bone, which in turn is the parent of the proprot bone.
The Body bone is the parent of the Body main mesh, of the 4 Body Lods, the 2 collision objects and of all the hook objects. The proprot bone is the parent of its own proprot main mesh and its 2 Lods.

All the entities in the scene are editable mesh. All the bones objects and the root are constituted each of one face object with three vertices.
Hooks are exported as cubes.



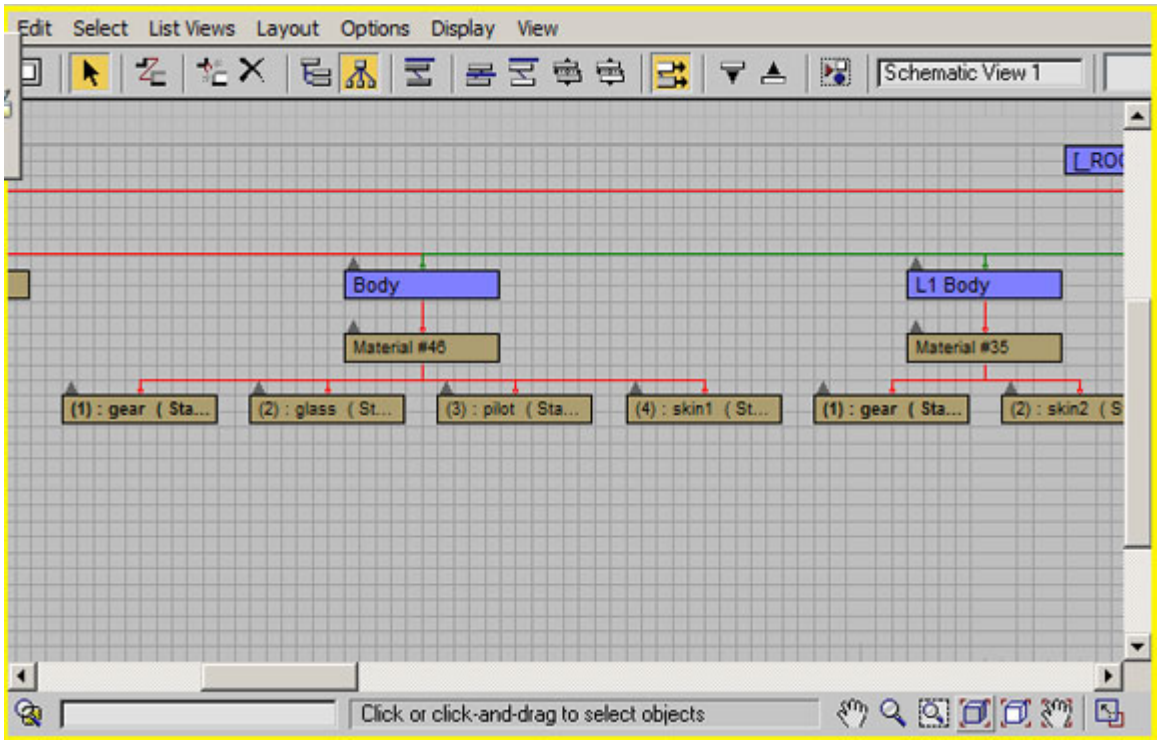
^ The _Engine1Smoke <BASE> Body HK hook selected

As you can see, all the hook objects have been positionned at the coordinates 0,0,0
The hooks also have a transformation matrix, stored inside the Body.msh file.
You will have to move the hooks to the correct position again, if you want to export the file again to Mshconverter (for example, you could have decided to create a Ju-87 G with two underwing BK37 37mm anti-tank gun). All you would need to do is create the 3D objects for the cannons and include them in the Body main mesh and each of the Lods (as simpler objects). Then you'd need to move the _MGUN01 and _MGUN02 hooks to the new position (the end exit of the barrel of the 37mm gun instead of the end exit of the wing MG17 machine gun). Of course, theses new objects should have a material (so they can be textured).



^ The L3 Body lod selected
Notice how the lower resolution Lods really have a lower poly count. They are extremely simple objects and don't need at all to be detailled, as the camera will be so far from them when they're displayed, that we wont notice the lack of detail. Notice also how the size of the various parts have been increased. The airplane would be displayed as only a few pixels on screen at this distance. The parts needs to be a little bigger, otherwise you might not even notice the plane at this distance.

The materials:



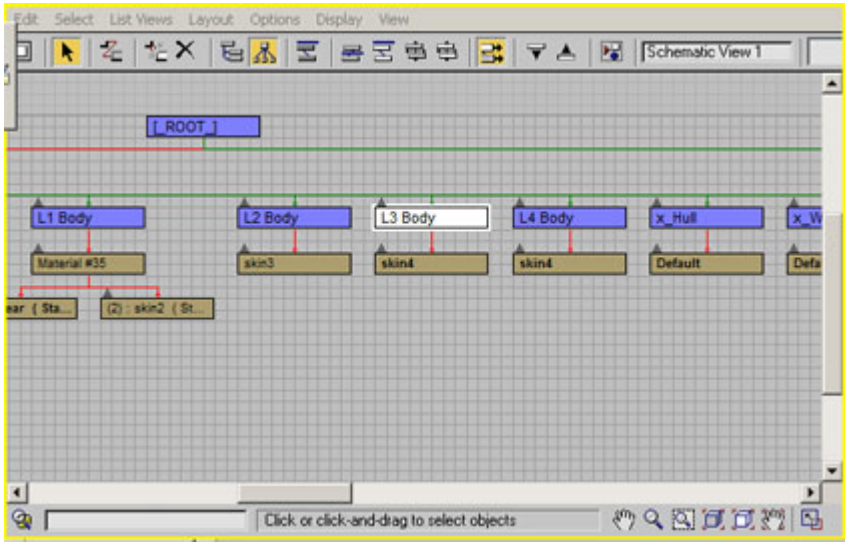
^ The materials for the Body main mesh and L1 Body LOD

Notice that there are only materials applied to the objects, and there is no texture nodes. This is because only the material names are stored in the *.msh files. Theses materials names are linked to the *.mat files.
For example, il there is a material called « gear » here, there is a corresponding gear.mat file in the unit's 3dobj folder. Even the parameters of the materials in Max are not needed by the game engine: it will use the parameters stored in the *.mat files instead. The information for the textures (which texture a material will be using) is also stored in the *.mat file.

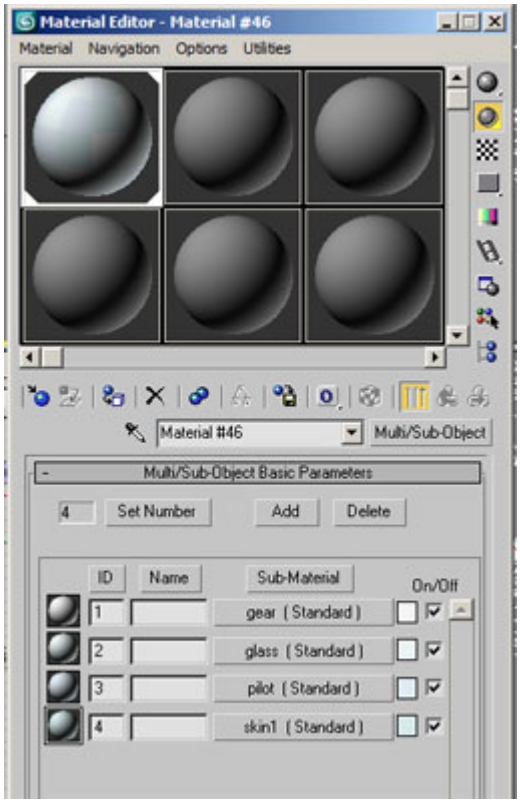
Notice that the Body main mesh and the L1 Body first Lod have a multi/sub-object material applied to them. In fact, it means that the object will use four different materials (and therefore, 4 different textures: one for the wheels, one for the cockpit glass, one for the pilot and gunner, and the main texture for the fuselage and wings of the Stuka. We can already understand how the materials are set according to which resolution Lod they're made for: here, the most detailled model that would be displayed on screen (the Body main mesh) has the most materials. The use of the material « glass » and the material « pilot » already tells us that the cockpit will be transparent, and that we'll be able to see the Stuka crew through the cockpit windows. For the L1 Body lod, that will be seen from further away, the transparent material will not be used for the cockpit, thus sparing the need to have the crew included in the L1 Lod and the load imposed on the game engine by having to display transparent parts. The main mesh Body (the Lod0) will use the skin1 material, and thus the corresponding skin1.dds texture (the highest resolution texture with a size of 1024x1024 pixels). The L1 Body Lod (the Lod1) will use skin2 material and the corresponding smaller texture half the size of the main mesh texture: skin2.DDS sized 512x512 pixels. Lod2 will use the skin3 material and texture (256x256 pixels), while Lod3 will use the skin4 material and texture (128x128 pixels). Lod4 will use the same material and texture than Lod3, but could have use a skin5 material with a skin5 texture in 64x64 pixels.

Only the Body main mesh and the Lod1 L1 Body are using a multi/sub-object material. Other

lower resolution Lods will be seen from too far away to justify the use of several materials. Only the skin3.DDS and skin4.DDS textures will be used, and then the material will be a simpler one:



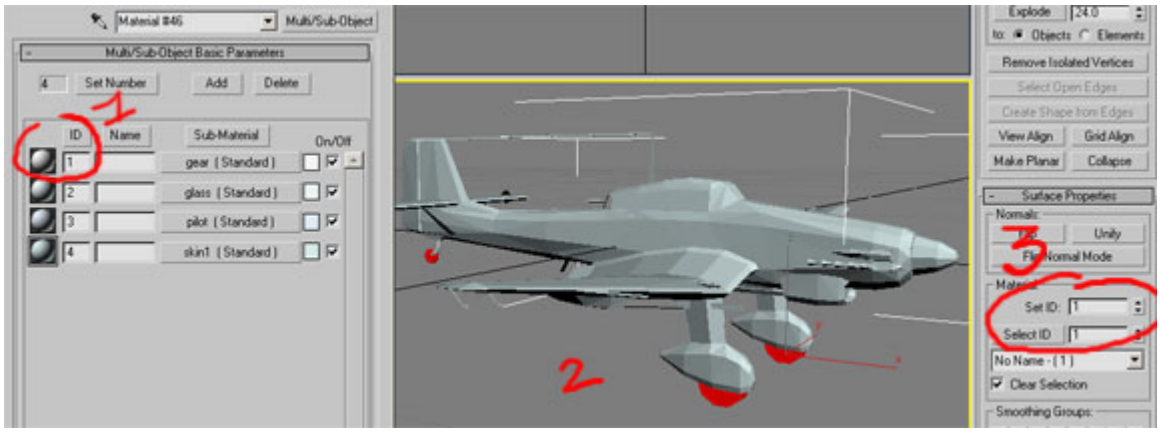
^ the L3 Body lod with the simpler material skin4



^ The Body main mesh multi/sub-object material Material #46

A multi/sub-object material means that some polygons of the model will have a material assigned to them, while some other polygons will have the other materials assigned to them.
Each polygons group (or FaceGroup) having the same material to them will have the same ID. here, all the polygons assigned with the « gear » material will have ID 1, all polygons assigned with the « glass » material will have ID 2, all polygons assigned with the « pilot » material will have ID 3, all polygons assigned with the « skin1 » material will have ID 4. This is not mandatory to have a multi/sub-object material: you could have only one skin1material assigned to the whole

mesh, provided that the gear, pilot and glass images are part of the skin1.DDS texture. But in the case of the Stuka, a multi/sub-object material have been assigned because it allows to reuse textures on several other units in the game (for example, all the pilots in all the german planes will be assigned this same pilot texture).

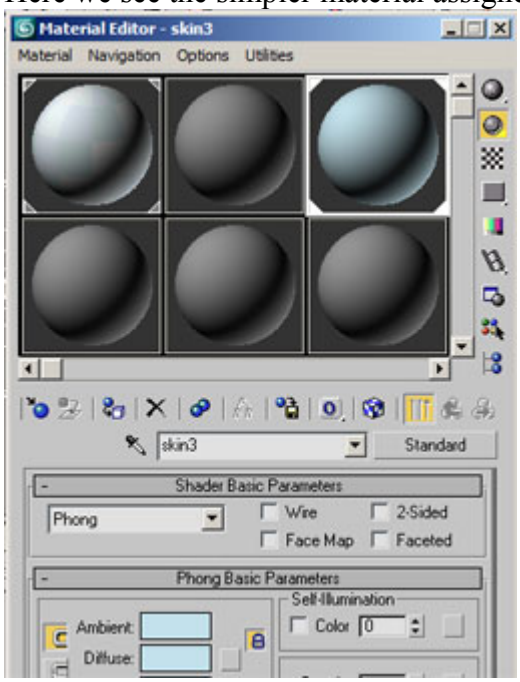


^The polys that have been assigned the ID 1 « gear » material on the Stuka.

(1) the « gear » material with ID 1 in the Material Editor.
(2) The FaceGroup with all the ID 1 material selected
(3) Here you can select or assign a material ID to polygons of the model.

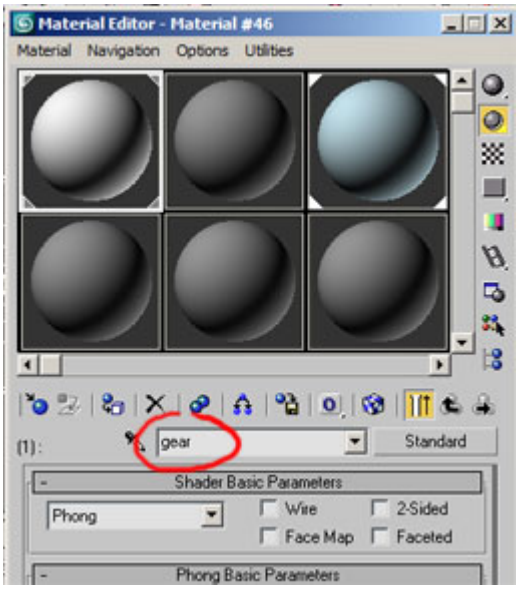
Note in (2) that the smoothing groups have not been carried through the conversion to 3DS format by MshConverter; that means you'll have to redo the smoothing groups if you want to export the model back into ToW in order to have the shading on the model to look correct (here, it is faceted instead of smooth).

Here we see the simpler material assigned to the Lod2 mesh L2 Body:



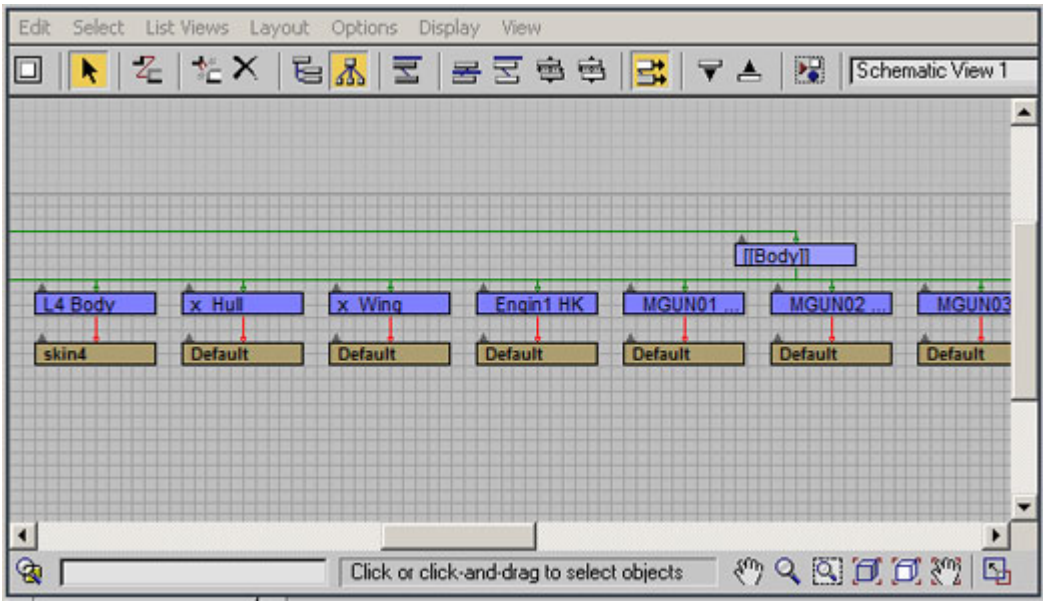
This simpler material (which is made of only one material and is assigned a single texture) does not have a material ID, as all the polys will be assigned the same material.
Note that the name of the material is very important, as it allows MshConverter and the game engine to know which *.mat file to use to display the materials and corresponding texture in game. For the L2 Body mesh, the mat file that will be used is skin3.mat (and this mat file will tell the game engine that the texture to be displayed is the skin3.DDS image).

For the multi/sub-object materials, you'll have to get inside each material to see or name the material that will call the corresponding mat file in game:



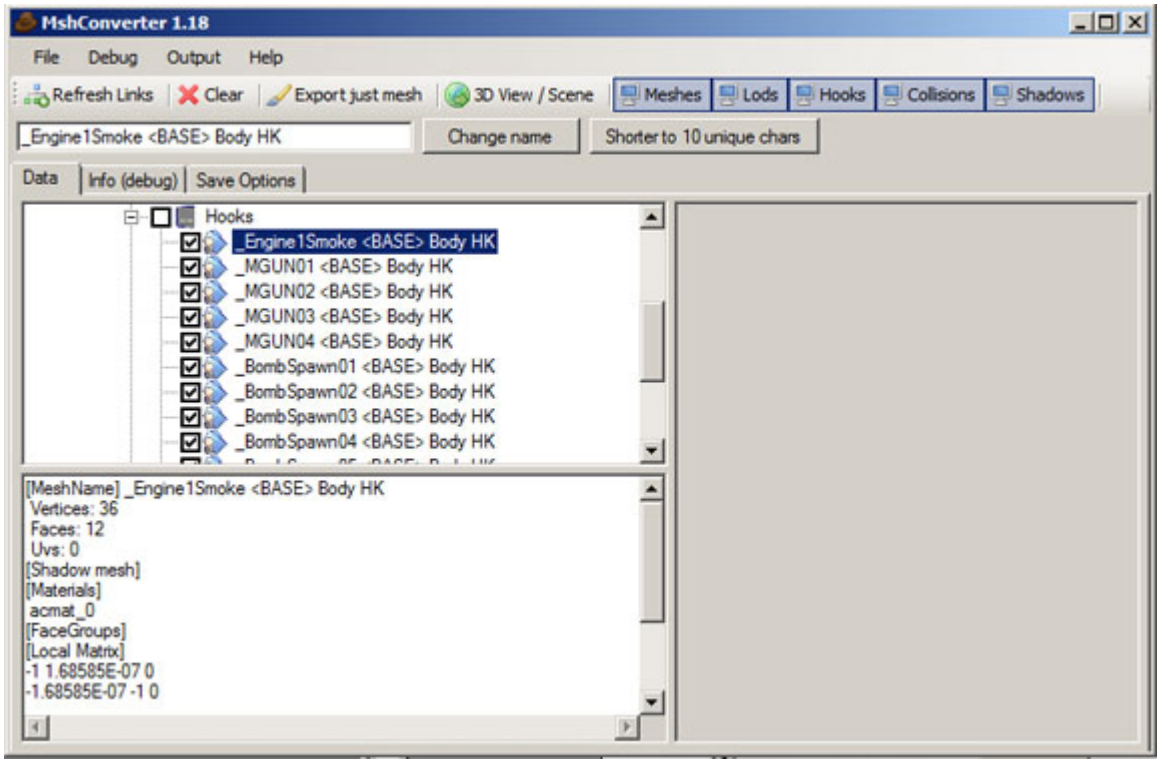
^The « gear » part of the multi/sub-object material called Material #46 for the Lod0 main mesh Body.
Note that you don't have to care about the name of the multi/sub-object material (Material #46), as it is not used by the game engine. Only the names of the sub-materials are important: gear, glass, pilot, and skin1.

There is something still a bit unclear to me about the materials assigned to the other objects in the scene:
3DS files saved with MshConverter are using the « Default » material for the bone objects and the hook objects



^ The hook objects have the Default material assigned in the Stuka 3DS file.

But if we look at the information about the hooks in MshConverter, we see that they should have a material called « acmat_0 » (see the picture underneath)



We can see in the lower information window of MshConverter, that the hook object _Engine1Smoke has a « acmat_0 » material assigned to it.
I have no idea if this material is really needed, nor what is its use. I suppose it is a material created by MshConverter. To be on the safe side, I've used this acmat_0 material on my Typhoon instead of the default material created by MshConverter, and the file is working in game. If you decide to use the hook object 3DS file located in the ressource folder of MshConverter, the hook have this acmat_0 material already assigned.

The bones objects have also been assigned with the « Default » material by MshConverter. We have no way of knowing thru MshConverter if they indeed have this material assigned, or even if they have one, as the information window contains only the transformation matrix information. Perhaps this is not even relevant, and the game engine does'nt care at all.

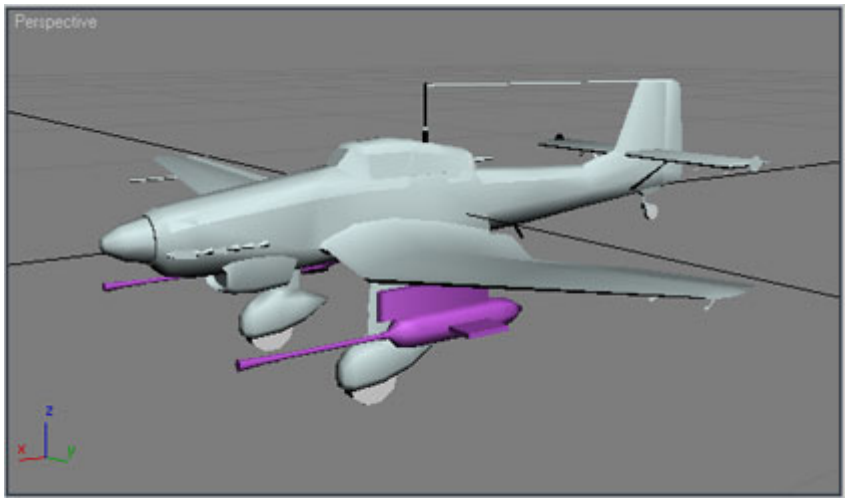
For the Collision objects, MshConverter seems to indicate that they are assigned no material.

Looking at the *.max files provided by 1C of ToW1 M4 and M6, we see that some collision objects have a «17 - Default » material assigned (the xHull hook amongst others), while the hook objects are assigned a variety of materials (_Hook_, Material #25, Material #26); This seems to indicate that the name for theses material is not very important, and that they are probably not used by the game engine. Probably only the materials referenced inside the *.msh files are important. The other could probably have whatever name you want.

One important thing about the materials and their names, is that each material referenced in the *.msh file must have a corresponding *.mat file in the 3dobj folder of the unit.
If this is not the case, the mesh will not appear in game.

Modifying the model:

Let's say we want to create a new version of the Junker Ju-87: we're going to make a new Ju-87 G-1 , the anti-tank version of the Stuka based on the Ju-87D-3.
We start by deleting all the stuff that was removed from the G-1 version: the dive brakes, the supports for the sirens, the wings bombs racks and the belly bomb attachement.
Then we create new objects for the two underwing BK3,7 cannons. You'll have to do this for each Lod, not only for the main mesh.



I suppose we could create a new bone containing the cannons, and attach this new bone to the remainder of the hierarchy by editing the hier.him file. I don't know for sure if this method would work, as there are no other planes in the game setup in the same way. In theory, it should work. We would have to assign a new material to the BK3,7 object, and create a corresponding *.mat file and *.DDS texture. Of course, we have to create not only the main mesh for the cannons, but also several Lods. I don't think we'll need four lods like for the Body. We could probably settle for two Lods and create 3 Lods at the most. We would not need to create collision objects for the guns, as the collisions for the planes are fairly simple.
Each of the Lods for the guns would need its own material, and its own texture.

An other method would be to include the polys of the cannons to the Body main mesh and to the Body lods. It means that we'll have to assign a new ID and material to the polys of the guns in the Body main mesh.
We could create a new sub-object material in the Material #46 with the ID 5, call the material « Guns1 » and assign it to the cannons polygons.
We'll the have the following materials in the Body main mesh:

ID 1 = gear
ID 2 = glass
ID 3 = pilot
ID 4 = skin1
ID 5 = guns1

Next in the same way, we'll need to create a new sub-object material in the Material #35 of the L1 Body lod. We'll then have the following material setup for the L1 Body lod:

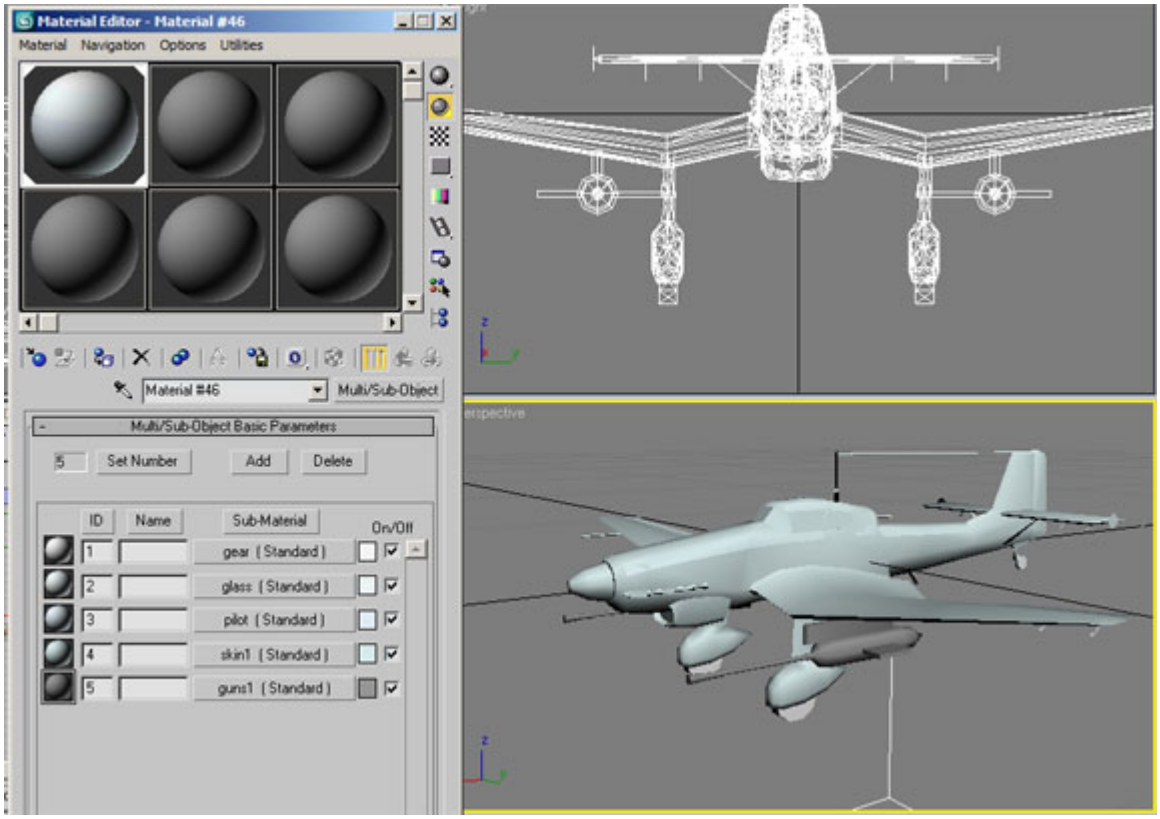
ID 1 = gear
ID 2 = skin2
ID 3 = guns2

For the L2 Body lod, we will change the simple material for a Multi/sub-object material:

ID 1 = skin3
ID 2 = guns3

We'll have of course to texture the new cannons object, and create UVs for them.

In the end, we'll have created several gunsX.mat files and the corresponding textures (they don't have to be called gunsX.DDS, but you can give the textures the same names as the materials for more easily remembering which mat file correspond to which texture).



^ The new sub-object material « guns1 » added to the Material #46 of the Body main mesh, and assigned to the polygons of the BK37 gondola cannons.

Once we've finished setting up the materials for Body, L1 Body and L2 Body and added the cannons objects for the guns to the lower resolution Lods (The poly count of the new guns should of course be simplified according to each corresponding Lod), we're ready to export the model back to MshConverter. Simply make sure before that, to have repositionned all the hook objects at their correct positions. You can ignore the _BombSpawn hooks, as we're not going to use them. No need to delete them, you can keep them. But since the bombs will not be functional in our final model, we don't really need them at the correct spot. Take special care of the positioning of the hooks _MGUN01 and _MGUN02, as we're going to use them for the two BK37 guns. Simply move each hook at the end of the barrel of the cannon, so that smoke and flames will be generated at the right spot.
We'll need to export our scene in *.3DS format.

Make sure first to delete the _ROOT_object (we'll not need it). We'll save the 3DS file with the name _ROOT_.3DS and when opening the file in MshConverter, the application will recreate the node _ROOT_ automatically. To do this, disconnect the [Body] bone from the _ROOT_ bone then delete the root.

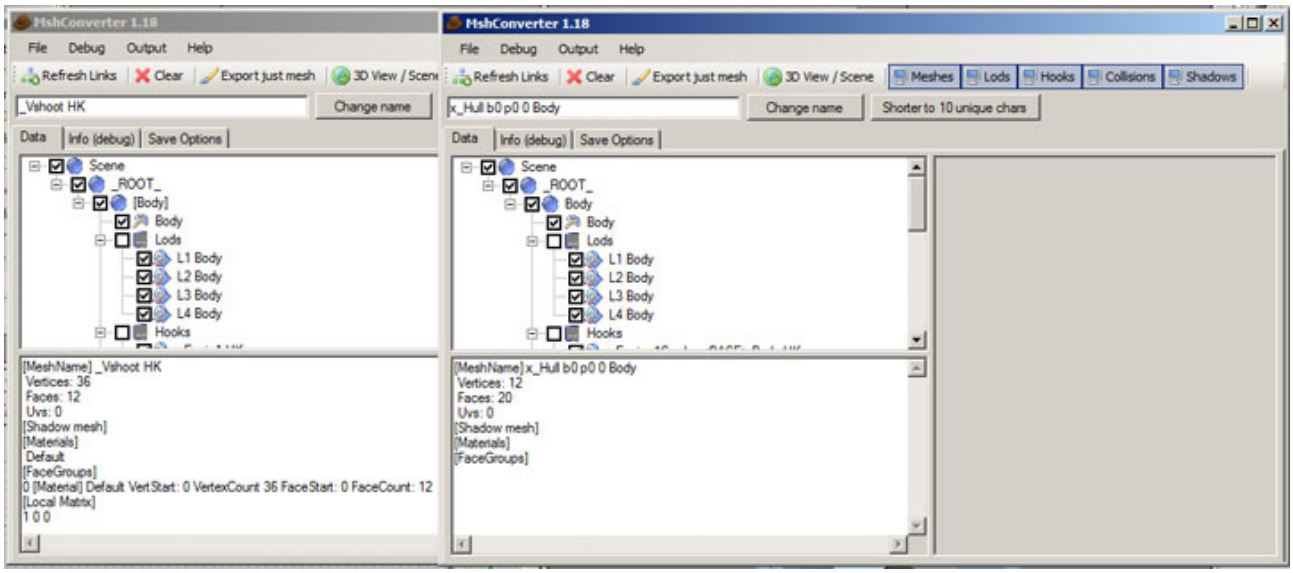
Export your scene as 3DS format and give it the name _ROOT_.3DS

Preparing your model for export to ToW with MshConverter:

Open the _ROOT_.3DS file you just saved in MshConverter.

The first task at hand is to rename all the objects so they have the correct names needed to export the Stuka model to the format needed for ToW. We'll save the hierarchy as a hier.him file, and MshConverter will create the hier.him file automatically, as well as saving all the *.msh file that constitute the model. For MshConverter to handle the conversion correctly, we need to give all the entities the same names they had before we shortened the names to 10 characters.

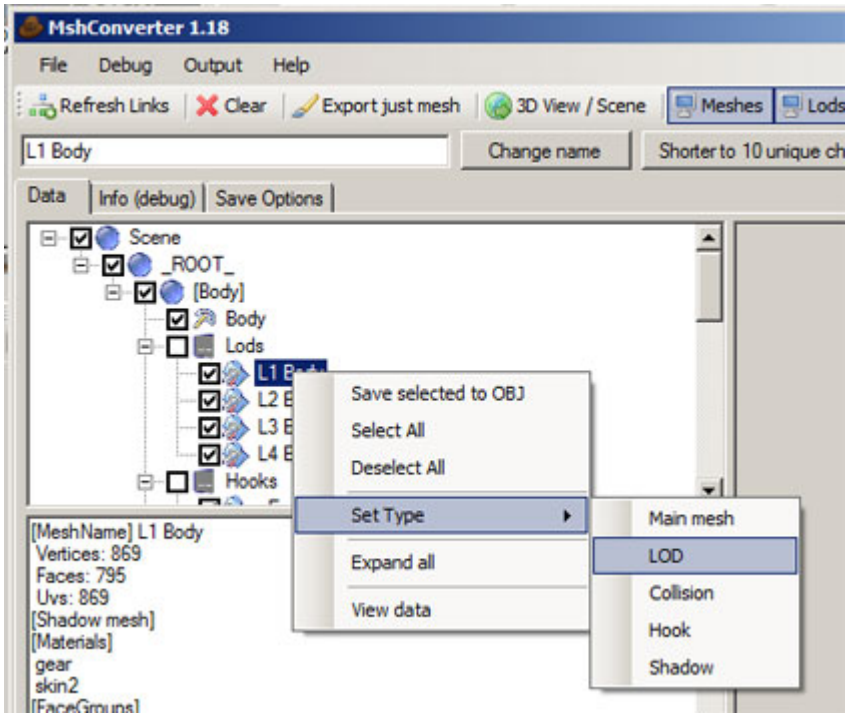
The easiest way to do it is to open an other instance of MshConverter, and load the hier.him file of the Stuka located in the ToW_extracted_files folder.
Putting the two instances of MshConverter side by side will allow us to easily copy the names of the entities from one instance to the other one, and renaming the objects.



^ Two instances of MshConverter opened at the same time

This is the time to check if MshConverter has detected all the entities as the correct type. If you made a mistake in the length of names, or if you did'nt put the correct prefix or suffix for the objects, chances are that MshConverter will have interpreted some of the objects as if they where belonging to an other type. For example, if you forgot to have one of your hook names ending with the suffix HK, MshConverter will believe that this entity is not a hook, but a main mesh, and will display the entity in the main mesh category. The same thing will automatically happen for your main meshes if they have a name like x0y0_dam0 (a lot of the static objects in the game have one or several meshes with this kind of name): MshConverter will mistake this objects for being a collision object, because the name start with an “x” (the prefix that is normaly used to identify collision objects).
You can correct the problem by right clicking on the faulty object and entering the correct type in the pop-up menu that will appear. Once this is done, simply click on the “refresh links” buttons to have MshConverter reorganizing the hierarchy correctly.

If we followed the naming conventions properly, there should be no reason to correct the object types, as they will all have been correctly identified as the correct type by MshConverter.



^ the menu for setting correct type for the objects

Don't forget to hit the Refresh Links button after having changed the type of an object.

Once you're sure that the types of objects are set correctly, you can start renaming the object to their former names:

For the Stuka, you'll have to do it for the Body bone/group (to delete the “[ ]”) and for all the hooks and collision object that had a name way longer than 10 characters.

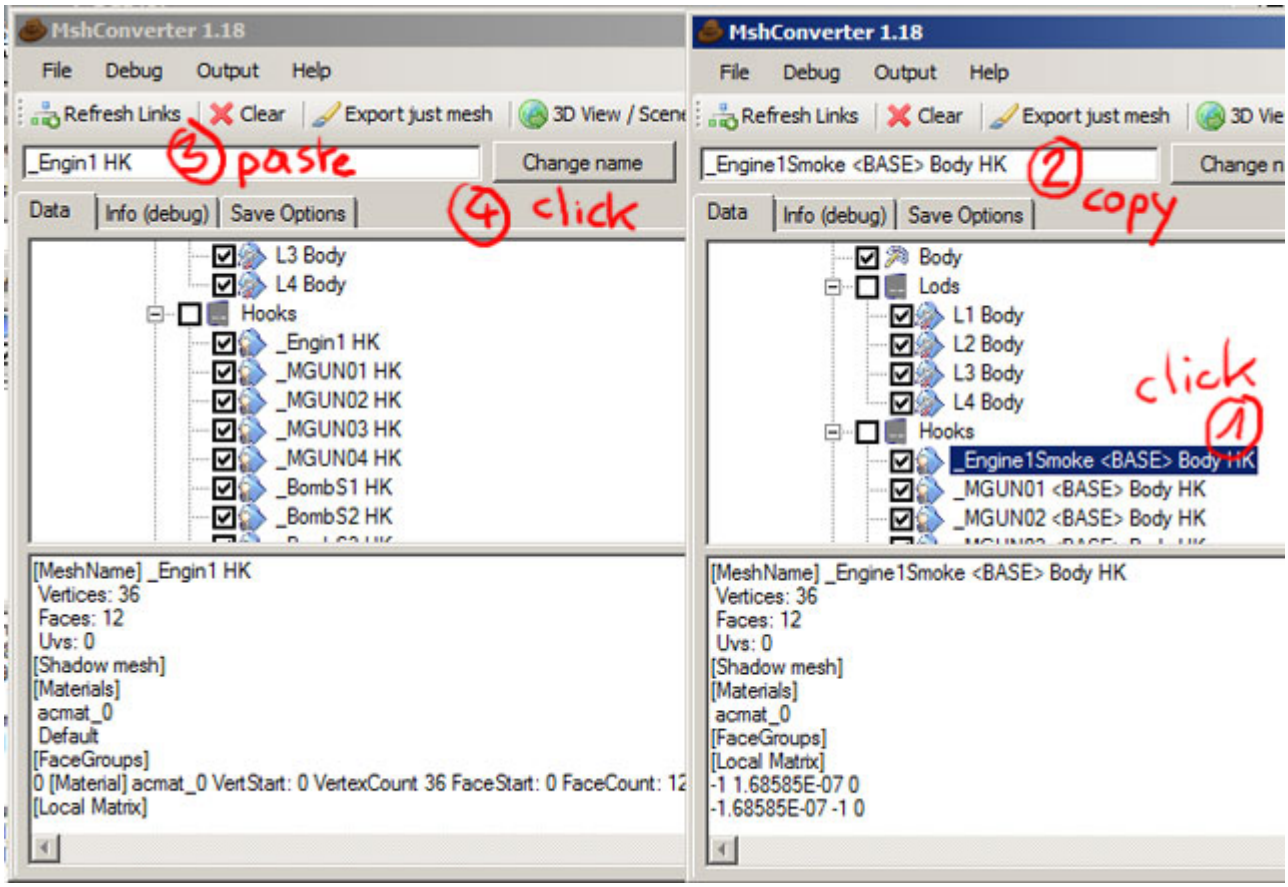
First select the entity with the shorter name you want to change in the first MshConverter instance.

Then in the second instance of MshConverter, select the same entity with the longer name. It's name will be displayed in the dialog bow on top.

Select the name in the dialog box and copy it.

Then in the first instance of MshConverter, paste the new longer name to replace the short name.

And finally, hit the “change name” button to modify and apply the new name



.If you look at the picture above, you'll notice that in the MshConverter instance to the right (displaying the files before we changed anything), the hook object Engine1Smoke have only one material, the acmat_0 material we mentionned before. Remember that inside 3DSmax, the hook objects did'nt have this material applied, but a “Default” material.

Now look at the left instance of MshConverter (displaying the modified 3DS file we just loaded). We see that now, MshConverter is displaying the -Engin1 HK hook as having two materials: the Default and acmat_0 materials. This is because I forgot to change the Default material name to acmat_0 inside Max. if I had done that, there wouldn't be any Default material showing, only he acmat_0 material would be there. In order to have the files similar to the state they were in before we modified anything, don't forget the rename the Default material to acmat_0 in Max.

Keep renaming all the hooks and the two collision shapes, as well as the proprot bone.

Make sure there is no mistakes in the names, or the conversion might create meshes that would not work as intended, or not work at all.
So we follow exactly the opposite process we have done when shortening the names. You'll have to modify the names like this:

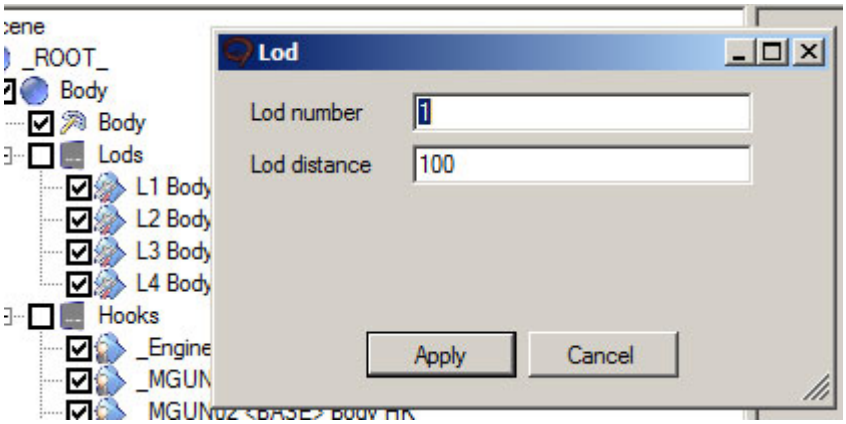
[Body] ----------> Body
_Engin1 HK ----------> _Engine1Smoke <BASE> Body HK
_MGUN01 HK----------> _MGUN01 <BASE> Body HK
_MGUN02 HK----------> _MGUN02 <BASE> Body HK
_MGUN03 HK----------> _MGUN03 <BASE> Body HK
_MGUN04 HK----------> _MGUN04 <BASE> Body HK
_BombS1 HK----------> _BombSpawn01 <BASE> Body HK
_BombS2 HK----------> _BombSpawn02 <BASE> Body HK
_BombS3 HK----------> _BombSpawn03 <BASE> Body HK
_BombS4 HK----------> _BombSpawn04 <BASE> Body HK
_BombS5 HK----------> _BombSpawn05 <BASE> Body HK
_BombS6 HK----------> _BombSpawn06 <BASE> Body HK
_BombS7 HK----------> _BombSpawn07 <BASE> Body HK
_BombSi HK----------> _BombsSign <BASE> Body HK
_Vshoot HK----------> _Vshooter_0 <BASE> Body HK
x_Hull----------> x_Hull b0 p0 0 Body
x_Wing----------> x_Wing b0 p1 0 Body
[proprot]----------> proprot

Once everything is renamed correctly, you're ready to set the visibility distance up for the Lods.

Setting up the Lods visibility distance:

You'll also need to open two instances of mshConverter: one with the hier.him file of the Stuka extracted from the SFS file, the other with our modified Stuka.

Double-click on the L1 Body Lod in the MshConverter instance where the original hier.him file is loaded. A pop-up windows open:

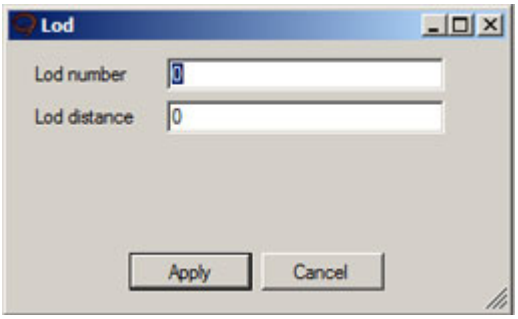


The lod number is the order in which the Lods will be displayed. L1 Body should have the number 1, while L2 will have number 2, L3 the number 3 … and so on.

The load distance is the distance in meter from the camera where the Lod will be made visible by

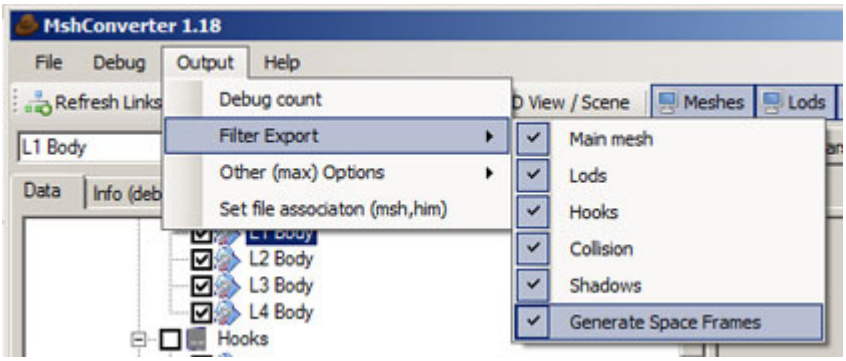
the game engine. Here it means that from 0m to 99m, Lod0 (Body main mesh) will be displayed. As soon as the camera is 100m away from the object, the L1 lod will be displayed, replacing Lod0

Once you know the Lod distance, double click the L1 Body lod in the instance of MshConverter who has our modified file opened.

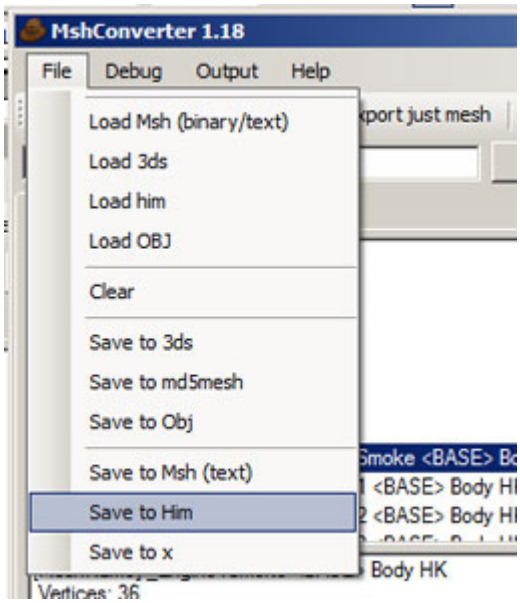


Replace the 0 by the appropriate 1 for the Lod number, and 0 by 100 for the Lod distance. Don't forget to do that, else the meshes will not display in game.
Follow the same procedure for all the other Lods in the Body bone, then do the same for all the Lods in the Proprot bone.

Once ou've finished, you're ready to convert to ToW format.
First, you need to change an option in the output menu to generate Space Frames (this option is used only for ToW meshes – for Il-2 meshes, this is not needed):



Save the hierarchy as a hier.him file



MshConverter will create a hier.him file with all the attaching information, as well as the transform matrixes for the bones and the hooks. It will also save one mesh file for the Body bone/group and one for the proprot bone/group. The Body.msh and proprot.msh files will be saved in msh/text format, which mean you'll be able to open the 3D model as a text file, and modify it if needs be.

This hier.him file will not work as is though, because we need to add the values for the VisibilitySphere, the CollisionObject sphere and the BoundBox, as MshConverter is not doing it automatically. Since we have simply modified the original Stuka, we will be able to use the same values as in the original hier.him file: our Ju-87 G-1 has the same size as the Ju-87 D-3, and so should have the exact same VisibilitySphere, the same CollisionObject sphere and the same Boundbow:

[_ROOT_]
VisibilitySphere 9.40860
CollisionObject sphere 9.40860 0.0 0.0 0.0
BoundBox -6.557688 -7.521715 -2.434510 5.548208 7.514601 2.457101

If we were creating a completely new object, we'll have to determine the radius of both sphere, as well as the coordinates of the Boundbox.
For the spheres, this is fairly easy: just create in your 3DSmax scene a sphere called Vsphere. This will be the visibilitySphere. Center it at coordinates 0,0,0 and increase the radius of the sphere till it is englobing all the model. This does'nt need to be very precise, some part of the models can intersect the sphere. The radius in meter of the sphere in max will be the radius of the VisibilitySphere in the hier.him file

The CollisionObject sphere as a radius usually slighly inferior to the VisibilitySphere. It should be centered at coordinates 0,0,0 too.

The BoundBox is the bounding box in which the model is contained enterely: it is the maximum length, width and height in meter of an imaginary box containing the model.
The coordinates correspond to the value in meter along the -X axis for the front of the model, the value along the -Y axis for the right side, the value along the -Z axis for the bottom of the model, the value along the +X axis for the back of the model, the value along the +Y axis for the left side of the model, and the value along the +Z axis for the top of the model.

Content of the msh (text) file:

We'll now have a look inside one of the *.msh/text files. Once converted to *.msh by MshConverter, the meshes of the Stuka will be saved in a text format instead of the binary format that was extracted from the SFS file. We can open this file with Notepad now, and have access to all the data of the 3D file. We can edit the file manually if we need.

Let's open the Body.msh first:

If you didn't convert it to msh/txt already, you can do it now. Open the Body.msh in MshConverter, and save it as a msh/text file. Name it Body.msh.txt (otherwise, you'll get an error message because you can't overwrite the file, since it is already opened in MshConverter)
Quit MshConverter. You can now rename the file to Body.msh if you want (but better keep both versions (the binary version as Body.msh, and the text version as Body.msh.txt)

So open the Body.msh.txt in Notepad:

//Generated by MshReader 1.18 by Dr.Jones

[Common]
NumBones 0
 FramesType Single
 NumFrames 1

I suppose the FramesType Single and NumFrames 1 entries indicate that this is not an animated mesh (there is only 1 frame)

[LOD]
100
100
200
400

These are the values set for the Lods view distance. I don't understand why L1 Body and L2 Body have the same value, though. For me it seems that both Lods would be displayed at the same time and disappear at the same time, which seems a bit strange. So far, I've seen this only in the case of airplanes. Usually, all Lods have a different view distance.

[Hooks]
_Engine1Smoke <BASE>
_MGUN01 <BASE>
_MGUN02 <BASE>
_MGUN03 <BASE>
_MGUN04 <BASE>
_BombSpawn01 <BASE>
_BombSpawn02 <BASE>
_BombSpawn03 <BASE>
_BombSpawn04 <BASE>
_BombSpawn05 <BASE>
_BombSpawn06 <BASE>
_BombSpawn07 <BASE>
_BombsSign <BASE>

_Vshooter_0 <BASE>

Here we have the list of all the hooks used by the Body.msh bone

[HookLoc]
-1 1.68585E-07 0 -1.68585E-07 -1 0 0 0 1 4.34024 0.031739 0.538859
1 0 0 0 1 0 0 0 1 2.87904 2.14848 -0.292161
1 0 0 0 1 0 0 0 1 2.87904 -2.19972 -0.292161
-1 1.68585E-07 0 -1.68585E-07 -1 0 0 0 1 -1.01846 -0.075256 0.826909
-1 1.68585E-07 0 -1.68585E-07 -1 0 0 0 1 -1.01846 0.027466 0.826909
1 0 0 0 1 0 0 0 1 1.87054 -0.0186067 -0.675631
1 0 0 0 1 0 0 0 1 1.90424 -4.79442 -0.400841
1 0 0 0 1 0 0 0 1 1.90424 -4.50242 -0.504781
1 0 0 0 1 0 0 0 1 1.90424 -4.19932 -0.423401
1 0 0 0 1 0 0 0 1 1.90424 4.18748 -0.453691
1 0 0 0 1 0 0 0 1 1.90424 4.50558 -0.482881
1 0 0 0 1 0 0 0 1 1.90424 4.77188 -0.306571
1 -3.37161E-07 0 3.37161E-07 1 0 0 0 1 1.68364 -0.0239254 -0.756621
1 0 0 0 1 0 0 0 1 2.39084 -0.0283058 1.03083

HookLoc is the list of the transformation matrix for each hook. You could edit the coordinates and orientations of the hooks here directly, without having to change the Max file and export again.

[Materials]
gear
glass
pilot
skin1

The list of materials used by the Body main mesh/ Lod0. There must be one *.mat file for each material in the unit 3dobj folder

[FaceGroups]
2135 1864
0 0 88 0 72 0
1 88 95 72 92 0
2 183 322 164 258 0
3 505 1630 422 1442 0
the first number (2135) is the number of vertices of the Body main mesh
Second number (1864) is the number of faces of the Body main mesh
Next lines: groups of faces with ID1 to ID4 for each materials
first material 0 gear vertex start 0, vertex count 88, FaceStart: 0 FaceCount: 72
second material 1 glass VertStart: 88 VertexCount 95 FaceStart: 72 FaceCount: 92
third material 2 pilot VertStart: 183 VertexCount 322 FaceStart: 164 FaceCount: 258
fourth material 3 skin1 VertStart: 505 VertexCount 1630 FaceStart: 422 FaceCount: 1442

[Vertices_Frame0]
2.258636 1.481171 -1.967926 0 -1 -0.000122
….and all the coordinates for the Body main mesh vertices

[Space_Frame0]

-1 0 0 1 0 0 0 -1 -0.000122
….and a bunch of coordinates for the Body main mesh

[MaterialMapping]
0.9294281 0.2491302
….and all the mapping coordinates for the Body main mesh

[Faces]
3 5 17
….and all the vertices number for the triangles of the Body main mesh

[LOD1_Materials]
gear
skin2
The materials for the L1 Body Lod

[LOD1_FaceGroups]
869 795
0 0 60 0 48 0
1 60 809 48 747 0

first number: number of vertices of L1 Body mesh
second number: number of faces of L1 Body mesh
second line: material 0 gear VertStart: 0 VertexCount 60 FaceStart: 0 FaceCount: 48
third line: material1 skin2 VertStart: 60 VertexCount 809 FaceStart: 48 FaceCount: 747

[LOD1_Vertices_Frame0]
2.244431 1.458572 -1.97171 0 -1 0
….and all the coordinates for the L1 Body Lod vertices

[LOD1_Space_Frame0]
-1 0 -0.0001609558 -0.8090527 0 -0.5877361 0 -1 0
….and a bunch of coordinates for the L1 Body mesh

[LOD1_MaterialMapping]
0.9459076 0.2464447
….and all the mapping coordinates for the L1 Body Lod

[LOD1_Faces]
13 11 1
The vertices numbers of the faces of L1 Body Lod

[LOD2_Materials]
skin3
The material assigned to the L2 Body Lod

[LOD2_FaceGroups]
280 236
0 0 280 0 236 0
First number (280)= number of vertices of L2 Body
second number (236)=number of faces of L2 Body
third line= material 0 skin3 VertStart: 0 VertexCount 280 FaceStart: 0 FaceCount: 236

[LOD2_Vertices_Frame0]
5.37764 -0.02557373 -0.2174072 0.7572021 0 -0.6531372
…. and all the other coordinates of the vertices of L2 Body

[LOD2_Space_Frame0]
A bunch of coordinates of L2 Body

[LOD2_MaterialMapping]
0.8730316 0.9374847
…. and all the other mapping coordinates for L2 Body

[LOD2_Faces]
0 7 5
… and all the other vertices numbers for L2 Body

[LOD3_Materials]
skin4
The material assigned to the L3 Body Lod

[LOD3_FaceGroups]
171 67
0 0 171 0 67 0
1**71= number of vertices of L3 Body**
67 = number of faces of L3 Body
last line: material 0 Skin 4 VertStart: 0 VertexCount 171 FaceStart: 0 FaceCount: 67

[LOD3_Vertices_Frame0]
The coordinates of the vertices of L3 Body

[LOD3_Space_Frame0]
Coordinates for L3 Body Lod

[LOD3_MaterialMapping]
mapping coordinates for L3 Body

[LOD3_Faces]
The vertice numbers of the faces of L3 Body

[LOD4_Materials]
skin4
The material assigned to L4 Body

[LOD4_FaceGroups]
54 26
0 0 54 0 26 0
54 = number of vertices
26 = number of faces
last line: material 0 skin4 VertStart: 0 VertexCount 54 FaceStart: 0 FaceCount: 26

[LOD4_Vertices_Frame0]

the vertices numbers of L4 Body

[LOD4_Space_Frame0]
Coordinates for L4 Body

[LOD4_MaterialMapping]
The mapping coordinates for L4 Body

[LOD4_Faces]
The vertice numbers of the faces of L4 body

[CoCommon]
NBlocks 1
This is the start of the collision section of the msh file.

[CoCommon_b0]
NParts 2
Npart 2 indicate that the collisions for the Body bone are made of two objects

[CoCommon_b0p0]
Type Mesh
NFrames 1
Name x_Hull
CoCommon_b0p0 give us the indication that this is the first collision (remember the first collision object name is always ending by b0 p0)
Nframes 1 may mean that the mesh is not animated as there is only one frame?
Name x_Hull = the name of the collision object.

[CoVer0_b0p0]
?

[CoNeiCnt_b0p0]
?

[CoNei_b0p0]
?

[CoFac_b0p0]
?

[CoCommon_b0p1]
Type Mesh
NFrames 1
Name x_Wing
the second collision object (name ending with b0 p1) called x_Wing

[CoVer0_b0p1]
?

[CoNeiCnt_b0p1]

?

[CoNei_b0p1]
?

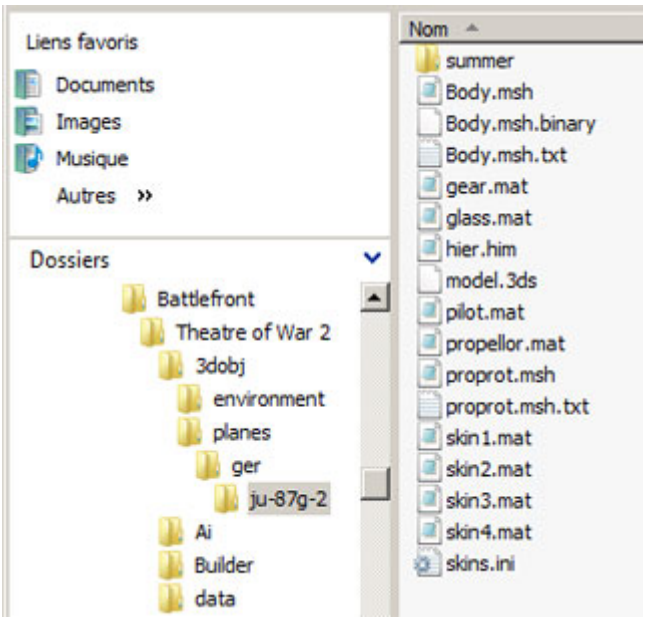
[CoFac_b0p1]
?

; eof

You can open the proprot.msh file in the same way and look at how it is organized.
The main difference will be that there is no hook objects, no collision and only two Lods.

There are a number of things you can modify on your 3D model without having to export it again to 3DSmax by editing either the Hier.him file or the*.msh files.
In theory, you can add new *.msh to the model by simply editing the hier.him file and adding a new bone and attaching it to your hierarchy and positioning it with the transformation matrix.
You can also change the materials referenced inside the *.msh file by editing them and changing the names. You could also switch one *.msh for an other (for example, to replace the gun of a tank by an other one)

Having a brand new model with all the 3D meshes, hier.him, *.mat file and the corresponding textures is only part of the work to make a functional model in ToW. We need to recreate all the folders and organize them in the same way as a similar object already in the game.
For example, we will have created a new folder, as well as sub-foldersin it inside the main ToW directory for our new Ju-87G-1, just like it is the case for the Ju-87D-3 inside the SFS file:



Here you can see how the new folders for the Ju-87G-1 should be organized. Note that the Body.msh.binary and Body.msh.txt are just back-up of the Body.msh file, one in the original binary format, the other in text format. You don't need them in the end to have your unit working in game. I just kept them as reference.

Textures and Mat files:

One thing to notice is that the skin1.mat to skin4.mat files shown in the picture above, inside the Ju-87G-1 folder, are not the mat files calling the fuselage and wings textures. They are all simply referencing a 16.tga texture (a 16x16 pixels yellow texture) located in the 3dobj.SFS with the following path: Theatre of War 2\3dobj\textures\. This texture is used when no other texture can be found in the unit folder (the unit will appear all textured in yellow)
The wings and fuselage camo texture is found in the Summer folder. This means you could have several different textures for your airplane: one for summer, and one for winter (for an airplane camouflaged in white during the winter months, for example). To enable this, you'd have to edit the skins.ini file and add a section [Skin_Winter1] in it, with the paths to the winter textures.
The 16.tga texture is not really a targa, it is a *.DDS texture with the extension changed to TGA. If you choose open as *.DDS, you should be able to open the file in photoshop

The other *.mat files like gear.mat, glass.mat and pilot.mat are the files used to tell the game engine which texture to display.
Let's open the gear.mat file in Notepad do see what's inside:

```
[ClassInfo]
  ClassName TMaterial
[General]
  tfDoubleSide 0
  tfShouldSort 1
  tfDropShadow 0
  tfGameTimer 1
[LightParams]
  Ambient 1.0
  Diffuse 1.0
  Specular 0.0
  SpecularPow 16
  Shine 0.0
[Layer0]
  TextureName ..\..\TEXTURES\gear.tga
  PaletteName
  Frame 0.0
  VisibleDistanceNear 0.0
  VisibleDistanceFar 10000.0
  TextureCoordScale 0.0 0.0 1.0 1.0
  ColorScale 1.0 1.0 1.0 1.0
  AlphaTestVal 0.0              // of [0.1, 0.5, 0.9]
  tfWrapX 1
  tfWrapY 1
  tfMinLinear 1
  tfMagLinear 1
  tfMipMap 1
  tfBlend 1
  tfBlendAdd 0
  tfTestA 1
  tfTestZ 1
  tfUpDateClear 0
```

tfModulate 1
tfNoTexture 0
tfAnimatePalette 0
tfAnimateSkippedFrames 0
tfNoWriteZ 1
tfDepthOffset 0
tfTranspBorder 0
tfTestZEqual 0
tfCompressMajorAlpha 1
tfNoCompress16Bit 1

You can see all the parameters of the material that the game engine will use to display the material: for example, tfDoubleSide 0 tells the game engine that the polygons having this material applied will not need to be rendered as a double face object (the texture will be seen only on one side, the side where the normal is pointing). TfDropShadow 0 for example will tell the game engine that the polys using this material will cast a shadow. TfDropShadow1 will make the polys cast no shadows.

This particular mat file has only the [Layer0] section because this material will use only one texture. Some other *.mat files sometime have two Layer sections: [Layer0] and [Layer1].
Layer 0 is used for the diffuse texture, and Layer 1 is used for the specular texture. A specular texture in ToW is a Normalmap texture. So objects using textures without normal maps have only the [Layer0] section, and thoses using also a Normal map will have both [Layer0] and [Layer1] section.
A mat file with a normal map will need the following parameters added to the [Layer1] section:
tfSpecularMap 1
tfBumpMap 1

Note that the [LightParams] section will have different parameters, depending wether the material will use a normalmap or not:

Material with specular

- *Ambient 1*
- *Diffuse 1*
- *Specular 0.3*
- *SpecularPow 32*
- *Shine 0*

Material without specular

- *Ambient 1*
- *Diffuse 1*
- *Specular 0*
- *SpecularPow 0* (sometime 16)
- *Shine 0*

transparent or semi transparent materials will require theses parameters to be set in the mat file:

for the [General] section: tfShouldSort 1

for the [layer0] section: tfBlend 1

Transparent materials (windshields or cockpits) for example, should be used only for the main mesh/Lod0 otherwise, to many of them could degrade performance. Beside, they will not be visible on lods seen from a greater distance, and would be a waste of ressources.

*.DDS textures: you should save them as [DXT3 ARGB 8 bpp | explicit alpha] in photoshop DDS plug-in. You have to generate the mipmaps at the same time you save the image, to allow ToW engine to use hardware texture mipmapping. To allow the mipmaps of the texture to be displayed in game, you have to enable the flag tfMipMap correctly for each layer in the mat file. Set it to tpMipMap 1 in most of the cases, except for the tank tracks, were the flag should be set to tpMipMap 0.

The first entry in the [Layer0] and [layer1] section defines which texture will be used by the material as well as its path: TextureName ..\..\TEXTURES\gear.tga for the gear.mat file.

You do not need to understand every parameters in the mat file to create a working one: usually, you'll copy an existing one and change only the texture name and path, perhaps the transparency flag, etc...

For a more detailled explanation, please refer to 1C documentation: “unit modeling guide for ToW”

Inside the Summer folder of the Stuka, we find the mat files for the regular color of the fuselage and wings: skin1.DDS to skin4.DDS

We have seen all the files dealing with the graphic aspect of our new model in ToW. But theses are not enough to have our unit fully working in game. We also have to edit and create a few text configuration files, so that the game engine will be able to acknoledge the presence of our new model, and activate its functionalities

The configuration files:

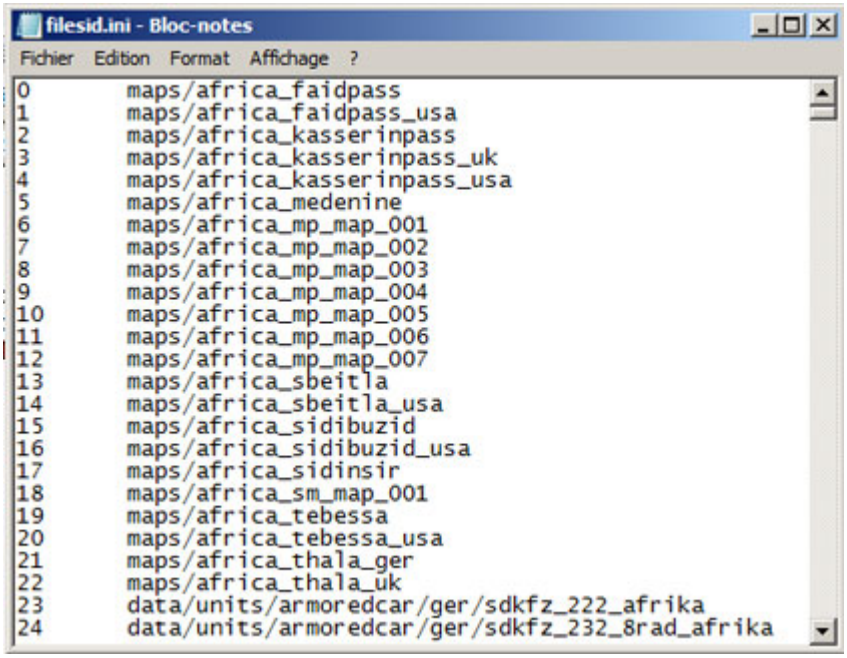
all the files dealing with the graphical aspect of our model were located inside the 3dobj folder. We'll need to have a look at the data folder now.

Have a look in the Data/settings folder found in the directory where we extracted all the files from the SFS files.

Filesid.ini:

Inside, you'll find a very important file: filesid.ini Each game element has a corresponding ID number that will be used in multiplayer when sending info through the net, instead of the full name and path of the element. Both client and server must have the same filesid.ini file. This could be a problem for modders as soon as someone have added a unit that other peoples don't have.

you can open this file with notepad



this file is simply a list of directories, textures, mat files, effect files, presets, sounds, with a corresponding ID number. Each ID number must be unique.

For exemple here, the ID number 0 is given to the Faid Pass map.

If you add a new unit to the game, you'll have to create a new unit ID and entry in the filesid.ini file.

To modify this file, you'll need to recreate the same data/settings directory inside the main ToW folder, and copy the filesid.ini inside it. Then you can add your new entry and ID in it and the game engine will use this file instead of the one stored inside the data.sfs file.

You'll need to add this line to your new filesid.ini (use the entries for the Ju-87d-3 as an example of what to do):

3993 data/units/planes/ger/ju-87G-1

Note that the ID 3992 was the last one in the filesid.ini. You have to increment this number for your new unit.

We'll have to had more entries in this file later, but we'll speak of this later in this document (ammo and magazines for the BK3,7 guns).

The next file found in the data/setting folder that we need to modify is the support.ini file

Support.ini:

This file is a list of all the assets available to the player as support (off map artillery and mortars, recon planes, fighters, bombers and ground attack planes).

We'll need to add our Ju-87G-1 as a new ground attack plane support.

Find the file support.ini located in the data/settings directory of the folder where you extracted all

the SFS files and copy it inside your newly created data/settings folder in the main ToW folder. The game engine will now use this file instead of the one stored in the data.SFS file. Open it to add the following lines:

[BattlePlane_Ger_Ju-87G-1_2units]

SupportType Attack

NumberOfCalls 5

NumberOfUnits 2

UnitName data/Units/Planes/Ger/Ju-87G-1

Delay 10

Timeout 60

Calldown 100

Formation 0 -100 150 20 // parentIndex rel_x rel_y rel_z

// SignalEffect

You've to choose a plane with a similar role as reference. So here, I've used the same settings as the Hs-129B attack plane instead of the Stuka dive bomber.

You could also add additional variations of support for the Ju-87G-1 (for example, here it will be only a two plane formation, but you can create an other section with 4 units if you so choose. Most of the data here speaks for itself. You can change the delay, time out and call down values and you can also modify the formation the planes will be flying (the position of the wingman relative to the leader). Thus, you could choose to have the airplanes flying in line abreast formation, in a finger four formation, or in line astern formation. They can be set to fly closer, or wider apart.

Units.utf8:

This file is located in the folder Data/local/en of the folder where you extracted the SFS files. Copy theses directories and files in your main ToW folder and start editing the units.utf8 file with wordpad.

This file is a text file listing all the names of the units in game

First name for a unit (on the left side) is the name that the game engine will use for the unit. Name to the right is the name that will be displayed on the GUI.

```
ArmoredCar.Ger.SdKfz_222_afrika     SdKfz 222 (reconnaissance armored car
mod.1935)

ArmoredCar.Ger.SdKfz_222_afrika.short     SdKfz 222

ArmoredCar.Ger.SdKfz_232_8Rad_afrika      SdKfz 232 (heavy armored car mod.1937)
ArmoredCar.Ger.SdKfz_232_8Rad_afrika.short      SdKfz 232
ArmoredCar.Ger.SdKfz_250_1_alt_afrika     SdKfz 250/1 (light half-track armored
```

```
car mod.1940)
.....

Planes.Ger.Ju-87D-3_afrika_1x500    Junkers Ju 87 D-3 (Junkers Ju 87D-3 Dive
Bomber, 1942)

.....
```

You have to add similar entries for your new G-1 Stuka :

Planes.Ger.Ju-87G-1 Junker Ju 87 G-1 (BK3,7 AT guns)

Then save the file.

Then don't forget that we have added a new weapon type in the game: the BK37 Anti Tank gun.

We'll need to add the description of this gun in the game, as well as of it's ammo.

The Bord Kanone 3,7 is based on the earlier 37 mm FlaK 18 37mm gun. We could use as a basis the FlaK 37mm gun that is used on the Sdkfz 6/2 : the 37 mm Flak 36 L/98 and change the data so that it matches the characteristics of the BK-3,7 gun.

We would do the same with the 37mm ammunition used by the Sdkfz 6/2, changing the type to APCR and matching the characteristics of the BK3,7 ammo. You could also create a second ammo *.ini file for the HE type of rounds used for ground attack in the BK3,7.

Since the Ju-87G-1 would be used mostly as an anti-tank flying platform, we'll decide to use only the APCR type of ammo in the gun, because it was not standard practice to mix the two kinds of ammo (APCR and HE in the magazine). I've found german data from 1944 about the mixing of ammo in the feed of the BK-37, but it indicates that for the antitank role, only APCR ammo with tracer was used. For ground attack, the BK3,7 gun magazine would be loaded with a mix of HE ammo: 2 HE rounds with tracer, 1 HE minengeschoss round with tracer, 1 HEI (incendiary) round with tracer. On widely known war time footage of Rudel shooting on russian boats, we see confirmation that all the HE rounds had a tracer, as we can easily follow the flight of each round toward its target.

Ammo *.ini:

This file must created in the Theater of War main directory, inside the folders data\ammo\ger

Following the procedures adopted for the files already in the game, we'll give our *.ini file a name containing the caliber as well as the name of the shell. This is not mandatory, but it is better to stick to this convention for ease of reference at a later time.

We create then a new text file named:

37mm_BK37_APCR.ini

content of the file: (you can use the 37mm_pzgr_40_apcr.ini as reference and modify only the data that you need)

You've to find information about the characteristics of the APCR ammunition of the BK3,7 gun:

I've found that the weight of the APCR projectile is 405g. So we'll use 0.405 for the Massa entry

You could also create new tracer, piercing and sound effects specific for the BK37, but we'll use effects that are already in the game here.

Note that the file content may be a bit different depending on which version of ToW you use.

[Base]

Name *37mm_BK37_APCR* Have to be the same as the folder name

[Properties]

AiAmmoType *APCR*

TraceMesh *Effects\Tracers\020_075mm_Red\mono.sim*

Kalibr *0.037*

Massa *0.355*

SolidPower *292*

SolidPowerRadius *1*

PierceEffects *PierceEffects_020-039mm*

PierceSounds *PierceEffects_020-039mm*

RicochetEffects *RicochetEffects_011_039mm*

RicochetSounds *RicochetEffects_020-039mm*

In the [properties] section *AiAmmoType* is for choosing which kind of ammo the AI will use. We could also add the HE type of ammo, as theses where also used with the BK3,7 canon. Will stick with only the APCR projectiles here for simplification of the process. Ideally, we should choose here all the types of ammo mounted in alternance in the ammobelt. Repartition of the rounds is handled by the corresponding magazine unit.ini file.

TraceMesh is for choosing which tracer visual effect to use. For ToW2, the line should read: *TraceMesh Effects\Tracers\020_075mm_Red\mono.sim* while for ToW1, it might be a bit different

(the same ammo might not be existing in ToW1) if this is a case, just choose the most similar file.

Kalibr *0.037* is the caliber in meter

Massa *0.405* is the mass in kilogram (405g for the BK3,7 APCR ammo)

SolidPower *292* ammount of damage inflicted is the armor in pierced

SolidPowerRadius *1* i suppose this is the damage inflicted by the schock wave of the non-explosive shell (solid). SplashPower and SplashPowerRadius are used in case of explosive ammo

PierceEffects *PierceEffects_020-039mm* This relate to the piercing visual effect preset.

PierceSounds *PierceEffects_020-039mm* This relate to the piercing sound effect preset.

RicochetEffects *RicochetEffects_011_039mm* This relate to the ricochet visual effect preset.

Magazine *.ini:

Each ammo must have a corresponding magazine *.ini file that should be created in \data\Items\Magazines\<country>\<magazine name>\unit.ini

for the BK3,7 it will be this file:

\data\Items\Magazines\ger\37mm_BK37_APCRx12\unit.ini

Note that i've added the number of rounds in the magazine, because different weapons might use the same round, but have a magazine with a higher or lower number of rounds. Or the same weapon could have different magazine with a higher or lower number of rounds. For the underwing gondola of the Stuka, the magazine has only twelve rounds.

Content of the unit.ini magazine file:

[Base]

Class *MAGAZINE*

IconName *Magazines\37mm_BK_37_APCRx12.mat* The mat file for displaying the ammo icon (you could have used the same picture than the 6 round *37mm_pzgr_40_APCR* magazine already in game). If not, you'll have to create a specific *.mat file and texture for your magazine.

Name *37mm_BK_37_APCRx12* Name of the magazine (same as the folder name)

[MoveType]

Class *FLY*

[Properties]

BulletPattern *Ger\37mm_pzgr_40_APCR* Which ammo will be used. If you want, you can have several kinds of ammo mounted on a belt in the magazine in alternance (for example, APCR,AT,HE,HE,APCR,AT,HE,HE... simply add each ammo type separated by a space. Here, the magazine will use only APCR type of ammunition.

Capacity *12* Number of rounds contained in the magazine (12 rnds for the BK3,7 of the stuka)

TracerIndex *1* Means that all shells will have a tracer (1)

Gun *.ini:

The guns *.ini file should be created in data/Guns/ger/

So we'll create a file called *37mm_BK_37.ini*

Content of the file: (use a similar gun as a basis) we'll use the 37mm_pak_35_36.ini

[Base]

Country *Ger*

Name *37mm_BK_37* Same as the *.ini name

[Properties]

Kalibr *0.037* Caliber in meter

useHookAsRel *False* I would have thought that this is set as *False* because this weapon is really a gun. Rockets and bombs probably would have the flag set as *True.* But this is not the case because bombs also have this flag set as False, like all the other guns I've checked. So I really don't know what this is for.

EffectPreset *020_039mm_Front* which visual shot effect preset the gun will use when fired

Sound *weapons.ger.37mm_pak_35_36* which shot sound effect preset the gun will use when fired

EffectMode *Auto* (could be *single* if the gun add to be reloaded manually between each shot)

ReloadTime *2 2* Min and Max reload time. Here, it is the same, because the gun is not reloaded manually. A BK3,7 gun as a rate of fire of 160 rounds per minute. 160/60sec= 2.66 seconds

shotFreq *2.6* The rate of fire. A BK3,7 gun as a rate of fire of 160 rounds per minute. 160/60sec= 2.66 seconds

traceFreq *1* could be the frequency of the tracers (but this is set in magazine*.ini)

Magazines *Ger\37mm_BK_37_APCRx12* The magazine used by the gun

ammoUsed *Ger\37mm_BK_37_APCR* The ammunition used by the gun

AimDistance	*0*	*50*	*100*	*500*	*1000*	*1500*	*2000*	Scale of aiming distances
AimRadius	*0.1*	*0.3*	*0.8*	*3.5*	*5.0*	*7.5*	*15.0*	accuracy values of the gun (influenced by Shooter skill)

ShootHintTimeout *10000* For ToW2 only: time that the sound can be heard by the ennemy in milliseconds which serve them to know where the shot was fired, even in not in LOS

ShootHintRange *300* Distance at which the shot can be heard and serve as a hint to the ennemy for locating the unit even if not in LOS (ToW2 only)

ShootHintRadius *20* Probable radius of error of locating gun position from the sound of its shot. (ToW2)

[Ger\37mm_BK_37_APCR] Ammunition section (point to the ammo *.ini file)

Ammo *Ger\37mm_BK_37_APCR*

Speed *1140* the muzzle? velocity of the gun, initial speed of the round? In meter per second. The BK3,7 gun has a muzzle velocity of between 1,170 or 1140 to 770 m/s. We'll have to choose one depending wether we're optimistic or pessimistic ;) or we'll choose an average. The data to be found is sometime widely different, depending on the source where you find it. Be specially critical of any information found on the internet. Here, I've not followed my advice , and settled my choice on an initial speed of 820m/s for the 640-645g HE round, and 762-770m/s for the 680/685g AP round. For the APCR round weighting 380g (or 405g , according to amore reliable source), the speed is supposed to be 1140m/s. I'll settle for 405g and 1140m/s

aimMinDist *10* The minimum distance from the target the gun will be fired

aimMaxDist *2000* The maximum distance from the target the gun will be fired. BK3,7 is supposed to have a usable range of 2600m. I don't know if the AI will use this distance as its max engagement distance though. So i'll not change anything right now, and will wait to see the AI use the guns in game to decide if the distance needs to be shortened (in case the AI have a tendency to fire the guns from to long a range, thus reducing accuracy). It could be that we need to reduce this distance substancially if it is used by the AI as its engagement distance, and reduce it to only a couple hundred meters.

HistMaxDist *6800* I suppose this is the “bullet history”, that is the amount of distance the game engine will “track” the flight of the round. It could also be the maximum range of the gun (well over the usable range where there is any chance to hit the target voluntarily)

Dispersion *1000 0.60* I suppose this means an average dispersion of 60cm for a 1000m distance.

LinesH	*0*	*100*	*500*	*1000*	*1500*	*2000*	*2500*	*3000*	*10000*	Target distance
Penetration	88	77	36	17	8	5	4	0	0	penetration in mm

Usually, it is hard to find penetration values at various ranges, unless you can find a document about tests carried during of after the war. For the BK3,7 APCR ammo, i've found only the penatration power at 100m: 130mm of plate steel (tests carried at Rechlin, probably at a 90° angle) So, this is too little info to claim any realism, but at least it gives us an estimation.

So i'll put 130mm at 100m in the penetration value, and estimate the other values, for lack of information. My intention is also to try to tweek the engagement range of the plane (if I can manage it) so that it will try to fire at 100m.

One way of finding an approximative value (if there is no other way to find the info), would be to draw a curve of the penetration values of the most similar ammo you can find, and modify the values of your gun according to this one. Here, i'll not bother doing this, and will keep the values of the 37mm Pak 40 APCR ammo, as this is only a tutorial :)

So, we're finished with creating the new armament for our new Stuka AT version.

There are other files we need to create to make our new unit functional in game:

Hier.ini, Parts.ini, Unit.ini

The data for a new unit is to be found in the data\Units\unittype\country\unit name directory.

For the G-1 Stuka, it will be: data\units\planes\ger\Ju-87g-1

you need to create three *.ini files in this directory: Unit.ini, Hier.ini and Parts.ini.

Hier.ini:

This is a file describing the hierarchy of the model: all the Bones and Hooks

[Root]

Type *UNIT*

[Hull]

Type *BONE*

Parent *Root* It means that the Bone Hull is linked to the ROOT

Bone *Body* This is the name of the mesh (Body.msh)

AxesOrientation *VHR* Orientation of the Bone

Min *+1* *+1* *+1*

Max *-1* *-1* *-1*

Speed *0* *0* *0*

Speed 0 0 0: It means that the Body mesh does'nt move relative to the ROOT parent.

[HookBombSign] This section relates to the BombSign Hook.

Type *HOOK*

Parent *Hull*

Hook *_BombsSign*

Other than that, I don't know what this hook is made for.

[HookBombSpawn1]

Type *HOOK*

Parent *Hull*

Hook *_BombSpawn01*

There is one such section for each _BombSpawn hook. This is not going to be used on our Stuka model, since it will not carry bombs. This is a remnant of the Ju-87D-3 model we've used to create the Ju-87G-1. None of the _BombSpawn sections have orientation values, as theses weapons can't be “aimed” by moving them individually, like a flexible MG, but are fixed relative to the airplane Hull

[HookBombSpawn2]

Type *HOOK*

Parent *Hull*

Hook *_BombSpawn02*

[HookBombSpawn3]

Type *HOOK*

Parent *Hull*

Hook *_BombSpawn03*

[HookBombSpawn4]

Type HOOK

Parent Hull

Hook _BombSpawn04

[HookBombSpawn5]

Type HOOK

Parent Hull

Hook _BombSpawn05

[HookBombSpawn6]

Type HOOK

Parent Hull

Hook _BombSpawn06

[HookBombSpawn7]

Type HOOK

Parent Hull

Hook _BombSpawn07

[HookVisible]

Type HOOK

Parent Hull

Hook _Vshooter_0

This is the Visibility sector of the pilot

[HookMGun1]

Type *HOOK*

Parent *Hull*

Hook *_MGUN01*

This is the Hook for one of the underwing gondola BK3,7 cannon. *_MGUN01 i*s the hook from where the bullets originate (used also for the flames and smoke effects)

[HookMGun2]

Type *HOOK*

Parent *Hull*

Hook *_MGUN02*

Note that here we have sections only for the main board weapons BK3,7 guns. There is no visibility sector for the gunner, not sections for the two rear-firing defensive MG-81Z _MGUN03 and _MGUN04 hooks. This seems to indicate that the rear gunner is not functionnal in game, and will not fire on an eventual pursuing fighter.

If they where to be functional, it seems logical there should be a visibility sector (and corresponding Hook in the model) for the rear gunner, as well as a [HookMGun3] for the _MGUN3 and _MGUN04 hooks, this section having the AxesOrientation entry with values different from 0 to define the firing sectors of the rear defensive flexible twin MG-81Z.

Parts.ini:

A big *ini file containing the damage model information, crew information, and armament information.

Content of the parts.ini file:

[Hull_Hull] The section Hull (as defined in the Hier.ini)

Type *Hull*

Name *Hull*

DamageType *Normal*

Collisions *x_Hull x_Wing* All the collision objects inside the Body.msh

AimCollisions *x_Hull* *T*he collision part that the ennemy will target

Health *800* The “hit points” of the part. When it reaches 0, part is destroyed

//Armor *Front Back Left Right Top Bottom*

Armor *2 2 2 2 2 2* Armor thickness in mm

External *True*

ArmorMTL *Steel* The game engine will treat this part as being make of steel. Could be Wood if the plane was constructed in wood, I suppose. Relates to the data/settings/damage.ini file.

IsBox *True*

isCritical *True* It means that if this part health reaches 0, the unit will be destroyed

[WeaponStatic_37mm_BK_37]

Type *WeaponStatic*

WeaponType *MachineGun* Weapon type for the AI (could perhaps better be set as Gun)

Variety *Primary* This indicate this is the primary weapon of the unit.

RatingType *Fly_Machine_Gun* Could probably have been better set as Gun. Will need testing in game to see if the AI is engaging only lightly armored or unarmored target because of the MG rating type.

Name *37mm_BK_37*

Parent *Hull*

Gun *Ger\37mm_BK_37* relates to the data/guns/ger/37mm_BK_37.ini file we created

SignNode *HookVisible*

SignType *Generic* a direct firing weapon that needs to have LOS to be fired.

Hooks *HookMGun1 HookMGun2* The two sections for the MG hooks in Hier.ini

ParkAngle *0*

DamageType *None* Means this can not be damaged. A tank gun will be set as Normal

External *True*

[WeaponStatic_Bomb] Will not be used by our G-1 as it has no bombs

Type *WeaponStatic*

WeaponType *Gun*

Variety *Twin*

RatingType *Heavy_Bomb*

Name *Bombs*

Parent *Hull*

Gun *Ger\Bombs_SC_500*

SignNode *HookVisible*

SignType *Bomb*

Hooks *HookBombSpawn1*

DamageType *None*

External *True*

[AeroControls_AeroControls] Relates to the maneuvrability of the plane.

Type *AeroControls*

Name *AeroControls*

MaxSpeed *80* I don't really know what the units are. Probably not kilometers per hour, as a Stuka can't fly at only 40 kph. Unit aiming precision seems to vary according to the speed set here.

MinSpeed *40*

MaxYawSpeed *20*

MaxAccel *15*

MaxPitchSpeed *50*

MaxRollSpeed *30*

PitchAccel *50*

MaxAccel *15*

YawAccel *15*

RollAccel *25*

SoundAero *planes.il2*

SoundRpmMin *4200*

SoundRpmMax *5900*

SoundRpmTime *5500*

[VehicleSkin_GerSummer1] texture sets used by the unit.

Type *VehicleSkin*

Name *Gerummer1* Name of the texture set

Skin *Summer1* Name of the texture set, as defined in the skins.ini file.

Division *GerEmpty* name of the regimental signs as defined in the skins.ini file.

Date1 *01.01.1939 31.12.1945*

This skin will be available during theses dates (if there are several texture sets like early war period, late war period.)

[SimpleAbilities_SimpleAbilities]

Type *SimpleAbilities*

Name *SimpleAbilities*

isTrailer *False*

isTractor *False*

isPassenger *False*

DestroyOnEmptyBackpack *False*

Unit.ini:

Content of the unit.ini file:

[Base]

Class *UNIT*

Classification *Planes*

Country *Ger*

Name *Ju-87G-1*

GuiType *Plane*

Mass *25500* This is supposed to be the weight in kilograms but this is set way too high for the Ju-87D-3 in the game. I don't even know if this is supposed to be the fully loaded weight with ammo, or without. Let's say it is the weight without ammo. It should be set to 5100kg or 6500kg, depending on the sources

Cost *3200* cost in command points

AIType *Plane* type for AI

VisibilityType *BattlePlane* visibility type for AI

RatingType *Plane* must have to do with how frightening to the ennemy the unit is

MeshName *3dobj\planes\Ger\Ju-87G-1_afrika\hier.him* Path to the hier.him file

DieEffect *Plane_Engine_Destroy* The flames and black smoke trailed by the plane when shot down.

Navigation *Plane*

Magnification *0.5*

[Dispatchers]

Script *plane.cpp* This is the AI script for the unit

[MoveType]

Class *Fly*

AbsoluteVertical *0*

UseGround *False*

[Speed]

TurnSpeed *0*

Filesid.ini:

in the data/settings/filesid.ini we had already added the following entry earlier:

3993 data/units/planes/ger/ju-87G-1

Now that we have created new ammo and magazine for the BK3,7 guns, we'll have to had them to the list.

3994 data/ammo/ger/37mm_BK_37_APCR

3995 data/item/magazines/ger/37mm_BK_37_APCRx12

If you've created also HE ammos for the BK3,7, you'll need to add them too

Guns.utf8:

Located in data/settings/en

the guns names are stored in this file:

you'll need to add the names of your new guns

```
Ger.37mm_BK_37    37mm BK3,7 (BordKanone 3,7cm)

Ger.37mm_BK_37.short    37mm BK3,7

There is a short and a long name for the gun
```

Items.utf8:

Located in data/settings/en

the magazine names are stored in this file:

you'll need to add the names of your new magazine

```
Magazine.Ger.37mm_BK_37_APCRx12.short     BK3,7 APCR

Magazine.Ger.37mm_BK_37_APCRx12     BK3,7 37mm \n Armor Piercing Composite Rigid
```

There is a line for the short name, and a line for the full name of the ammo

Once all this have been done, it is time to test your new unit in game. It should work if you've made no error in the configuration files, and of course provided that all the info i've written above is correct ;)