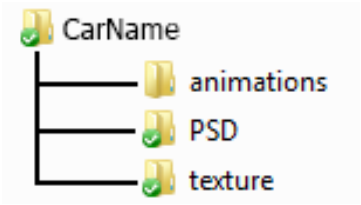
Car Pipeline 1.03 (work in progress)

Asset ORGANIZATION

When you create a new car for the editor, it is necessary to prepare a specific tree folder as follows:

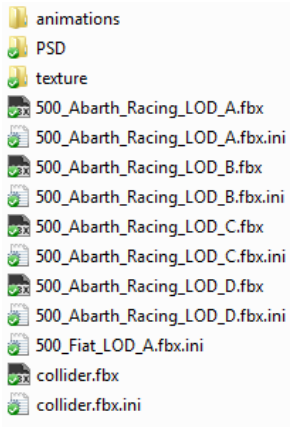


In the root of your CarName (where “car name” is the name of you car model, like Ferrari or bmw and so on…) the following 3 subfolders must be present:

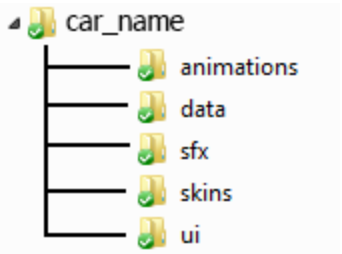
animations: This folder contains the FBX source files with animations and is also where the editor saves the animations clips.
PSD: This folder contains the PSD source files of the car textures that can be eventually modified.
texture: This folder contains the DDS textures.

NOTE: If the texture folder is missing, the car won’t be exported from the editor to the game.

After the folder creation, place your FBX file inside and the editor will read the asset from this folder to create the car package. Below is an example of all the FBX files of a car, placed inside the folder. Afterwards the editor has created the material files (*.ini) in the same folder.



The in game folder where you must place the **car.kn5** file is different from the one where the EDITOR reads the assets to create the proper package. This is the folder tree for in game cars:



The ROOT of car_name (your car title) contains:
1) all the **.kn5** files
2) the **driver_base_pos.knh** (a file that indicate the driver position in the car)
3) the textures called **tyre_0_shadow.png , tyre_1_shadow.png , tyre_2_shadow.png tyre_4_shadow.png** that are the wheel shadow textures, the **body_shadow.png** and the **logo.png**
4) The **animations** folder contains all the clip files *:ksanim of the driver and the car
5) The **data** folder contains all the configuration files, physics scripts and similar ini files
6) The **sfx** folder contains the sound samples and scripting files
7) The **skins** folder contains the car skins graphics and thumbnail previews for the interface
8) The **ui** folder contains thumbnails and scripts specific for the user interface

FILE FORMATS

TEXTURES

PNG (RGBA)
DDS (RGBA plus all the internal features of the common DDS plug-in for PHOTOSHOP. Indexed color DDS are not supported yet. **)**

3D file formats supported: **FBX** (plug in 2012.1 FBXSDK 7.2) (future version and updates will be tested)
XSI 2012 , 3DSmax 2012, Maya 2012 export in the above version.
For MODO, BLENDER, C4D and others, you must test their export and check if it works correctly in the editor.

DDS supported formats: to be defined. Coming soon.

3D MESHES

One set of **UV** coordinates is supported
In some shaders we add more layers but only increasing the tile of the existing UV set.

Mesh normals (normal can be tweaked as required, and it will be exported correctly)

Basic Shader: A texture must be assigned to the material.

NOTE: Every mesh MUST have TEXTURE UV coordinates.
The mesh must be (when possible) in quads. Do not triangulate the mesh if it is not necessary (look the image of the cockpit VHR to understand.)

Skinned mesh you can have as many bones as needed, but every single vertex can be influenced from up to 4 bones and not more

Vertex color is **NOT SUPPORTED**.

MESH PARTS OF A GENERIC CAR MODEL

The components of a car must be divided in many parts in order to manage animated objects, meshes and other features present in game. Here is a list of the optional and required meshes:

MAIN BODY **required** - must be present on LOD_A and LOD_B

DOORS optional - only on LOD A and if present on the model
on LOD_B the doors are not animated, and are welded on the chassis object.

MOTORHOOD **depends on car type** - if needed, must exist on LOD_A and LOD_B

FRONT BUMPER **depends on car type** - if needed, must exist on LOD_A and LOD_B

REAR BUMPER **depends on car type** - if needed, must exist on LOD_A and LOD_B

WHEEL HUB optional - contains the brake calipers

WHEEL RIM **required** - must exist on LOD_A and LOD_B. On LOD_C the wheels are simplified.

WHEEL RIM BLUR **required** - a version of the rim but with a texture blurred, must exist on LOD_A and LOD_B

WHEEL TYRE **required** - must exist on LOD_A and LOD_B. On LOD_C the wheels are simplified

BRAKE DISK **depends on car type** - if needed, must exist on LOD_A and LOD_B

FRONT LIGHT **depends on car type** - if needed, must exist on LOD_A and LOD_B and LOD_C

REAR LIGHT **depends on car type** - if needed, must exist on LOD_A and LOD_B and LOD_C

WIPERS **depends on car type** - if needed, must exist on LOD_A and LOD_B and LOD_C

FRONT WING **depends on car type** - if needed, must exist on LOD_A and LOD_B and LOD_C

REAR WING **depends on car type** - if needed, must exist on LOD_A and LOD_B and LOD_C

Cockpit mesh parts:

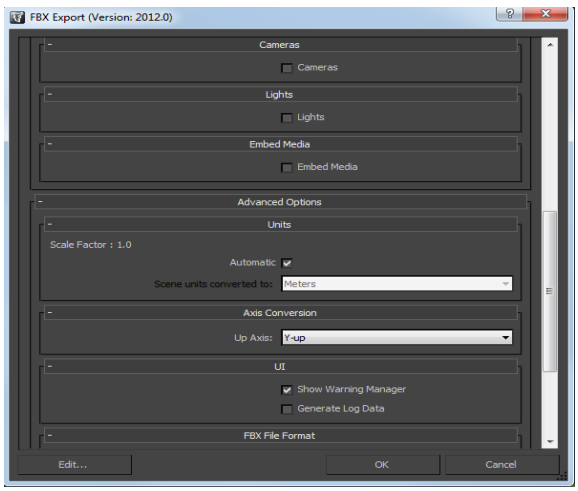
COCKPIT_HR **required** - high resolution cockpit, the one that you see when you drive

STEER **required** - steering wheel

STEER PADDLE **depends on car type** - those are the steering wheel gearchange paddles

SHIFT **depends on car type** - gear lever for manual or sequential gearbox

SECURITY BELTS **depends on car type** - if needed, must be present on LOD_A only. Explained later in the document



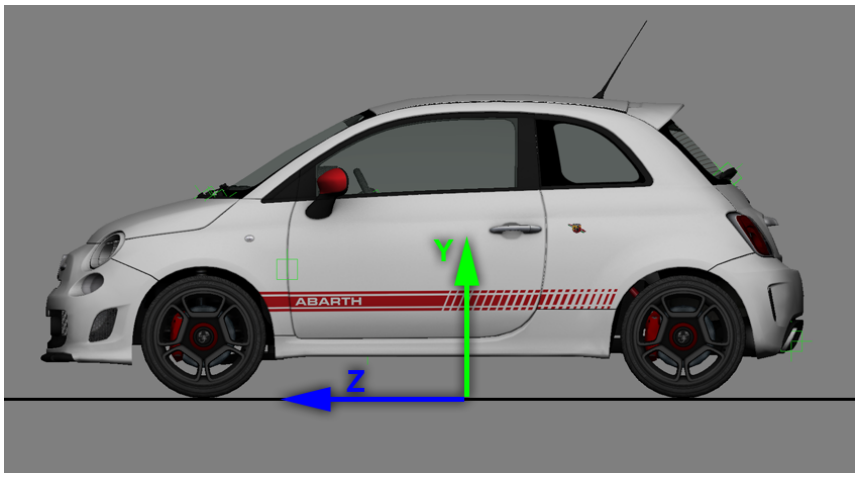
All the "**optional**" components can be present or not, depending of car.
An example scene is provided, and can be imported inside any application that supports FBX file format.
Programs that support FBX file format: MAYA 2012 3DS Max 2012 XSI 2012

NOTE:when exporting from 3ds max, always export with Y axis up.

SETUP THE CAR MODEL INSIDE 3D SPACE

The car must be oriented like in the image: The Z vector must be the front direction.
The model must be placed with the wheels on the Y 0 ground. (See image below)
The model bounding box must be centered in YXZ = 0.0.0. (See image below)
The car must have 4 different Levels Of Detail that have gradualy less polygon density.
(see DETAIL LEVELS Chapter)

The table below indicates the adviced vertex budget.
We encourage you to stay as close as possible to the proposed polygon numbers.



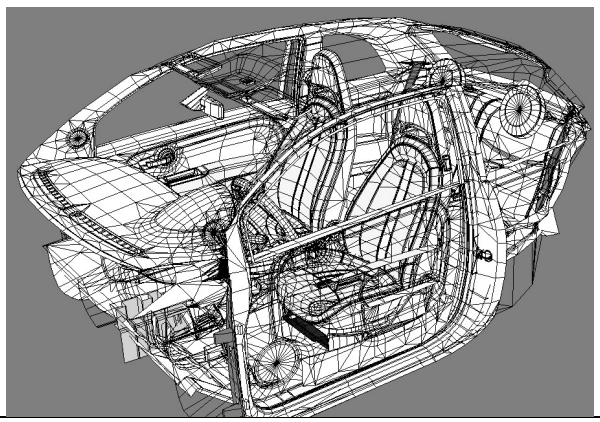
CAR BODY (wheels included) :

LOD A	**100.000**	**Polygons**
LOD B	**20.000**	**Polygons**
LOD C	**5.000**	**Polygons**
LOD D	**100**	**Polygons**

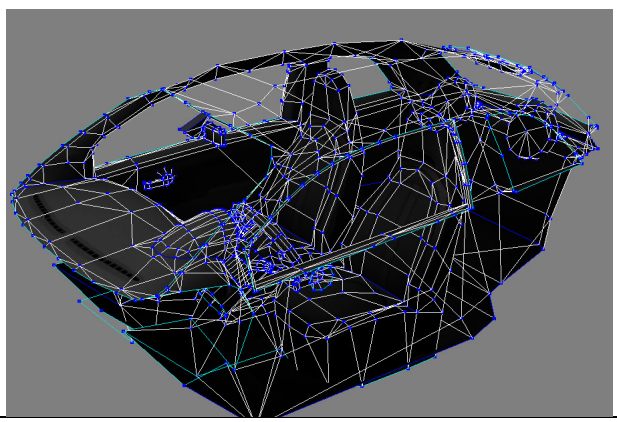
NOTE: Keep in mind that the LOD B will start to be visible at a distance of 15 meters or closer. If you create a very well made LOD B, you can reduce the distance of LOD A switch to LOD B.

The Cockpit has only 2 LODS, one very High Resolution (HR) for the **cockpit** camera and **showroom** view. Another Low Resolution (LR) LOD for exterior cameras, replays, and distant views.

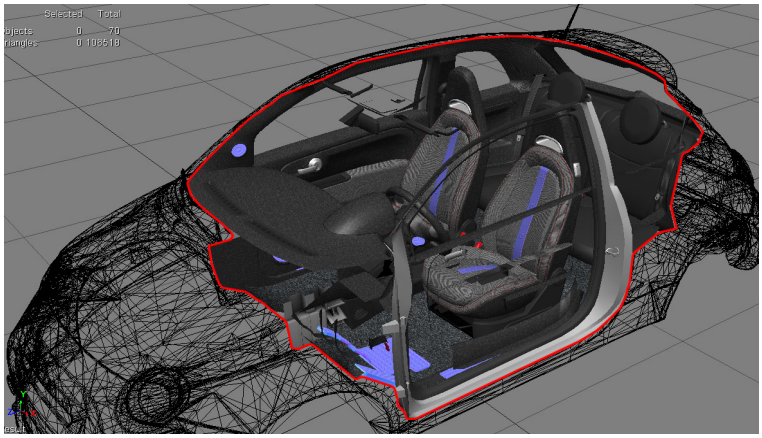
LOD A COCKPIT_HR 100.000 polygons
Cockpit HR LOD example



LOD A COCKPIT_LR 5000 polygons
Cockpit LR LOD example



NOTE: The cockpit HR and LR LODS must always fit the Body LOD_A because the EXTERIOR MESH, in game while driving, is the LOD_A mesh. When the camera moves far away, the cockpit LR will switch and you get a simplified version of the cockpit, with only one material and color very similar to the HR version.



“LODs” Level.of.details

LODs are a set of simplified models that change in relation of the camera distance.
This process is necessary in order to optimize the frame rate in game.
The LODs switch can be controlled from a script, named **lods.ini** placed in **AssettoCorsa/content/cars/CAR-NAME/data**. The script contains the following values:

```
[COCKPIT_HR]
DISTANCE_SWITCH=6 ;Indicates the distance (in meters) when the cockpit HR change to the cockpit LR (if present)
[LOD_0]
FILE=abarth500.kn5
IN=0
OUT=15 ;Indicates the distance (in meters) when lod_A changes with lod_B (if present)

[LOD_1]
FILE=abarth500_B.kn5
IN=15
OUT=30   ;Indicates the distance (in meters) when lod_B changes with lod_C (if present)

[LOD_2]
FILE=abarth500_C.kn5
IN=30
OUT=100  ;Indicates the distance (in meters) when lod_C changes with lod_D (if present)

[LOD_3]
FILE=abarth500_D.kn5
IN=100
OUT=600   ;Indicates the distance (in meters) when lod_D disappears from visual.
```

NOTE: Verify that the distance of LOD “out” match the “in” of the next lod, or your car will disappear while switching

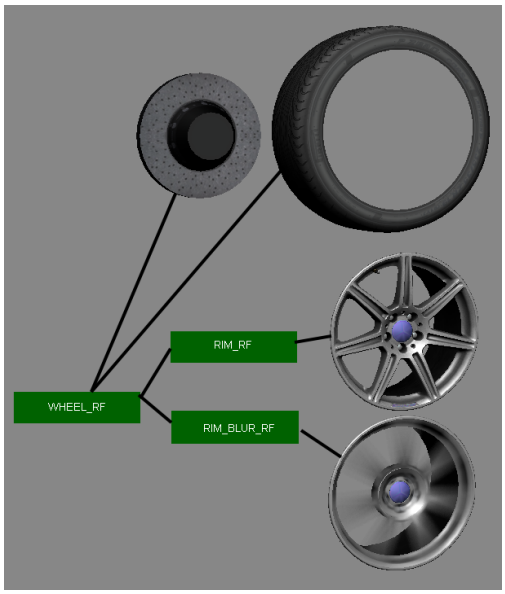
Additional info: The **LOD_B** must have the same null hierarchy as the **LOD_A** except for the **null** of the **COCKPIT_HR** that should not be present. Furthermore **null** of animated doors are not needed in **LOD_B**. Same as above for **LOD_C**. YOu can decide, if is no more visible, to remove also othe non ESSENTIAL nulls, liek wipers, wing.. and so on.

HIERARCHY and CENTERS

A template file called “**Template.fbx**” is provided as an example, showcasing how to setup a correct hierarchy for a car. The file contains a set of **NULLS** or **DUMMY** objects, that define the **CENTER** position of any piece of the car. The names of these **NULLS** must follow specific rules showcased below.
The **EDITOR** will recognize these essential **OBJECTS** in order to define the rotation pivot of the wheels, the suspensions and any animable object in the car.

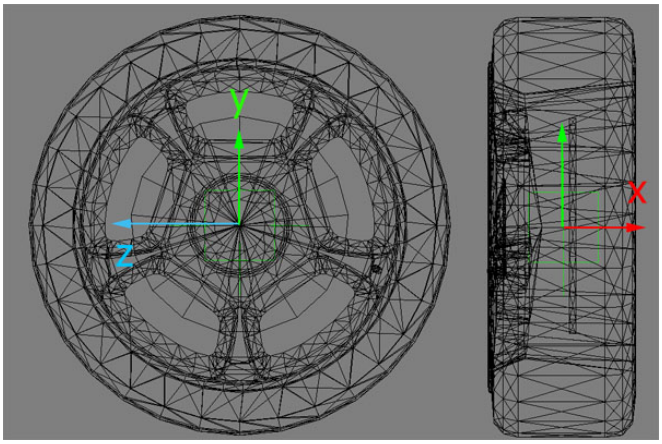
NOTE: Any object that is not a child of a NULL/ DUMMY will be managed like part of the CAR CHASSIS.
All the pieces of the geometry that belong to the car must be placed in a HIERARCHY to define the specific properties of each mesh object in the game.

Example image below: The pieces of the wheels are placed like children of the NULL of the WHEEL
In the same way you must place all the others pieces like children of the correct **NULL** that is designed for the



part that you are creating. So for the rim there is a dedicated **NULL** and so on for all the other parts. Remember that every **NULL** is also the **CENTER** of rotation. If your mesh is not properly placed under a **CHILD** with a correct center of rotation, the mesh will rotate wrongly.

See the example below : The geometry of the **WHEEL** is centered exactly on the **NULL**. This will permit a correct rotation.



In the car example file you'll be able to explore how we have placed all the **NULL's** and the relative geometry.

SUSPENSIONS Animation

The 3D suspensions of a car can be animated if needed.
In order to enable suspension animations, you have to edit the the script **"car.ini"** with the following values:
"USE_ANIMATED_SUSPENSIONS=1" to enable use of the animations
"USE_ANIMATED_SUSPENSIONS=0" to leave the animations as by default (disabled)
Usually this means the animations are disabled, but on some occasions they are always on like the steering wheel which is automatically animated.

NOTE: Animating suspensions do have some disadvantages. Animating suspensions follow predetermined arcs and movements, so the wheels do not rappresent visually the setup values chosen by the player in game. i.e. Camber angles might differ visibly from the values selected in setup screen.

We use a **NULL** hierarchy to animate the suspension geometry. Here's an example below

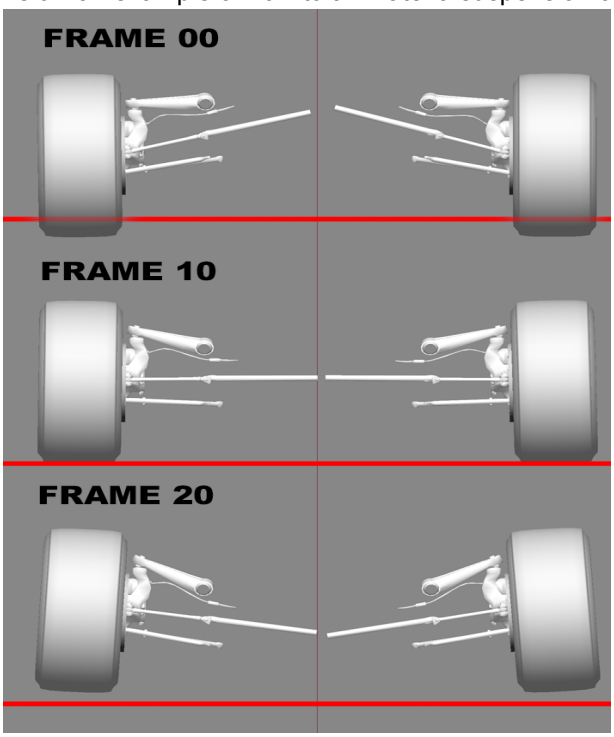
1) Set your timeline frames to (for example) 20 frames.
2) The frame 0 will be lower position.
3) The frame 10 will be the neutral position.
4) The frame 20 the higher position.

The engine works as follows: It verifies the position in the y axis of the suspension and finds the right frame to match the animation to the position of the physical suspension. It will interpolate the frames to generate a smooth movement. Assetto Corsa will search for the following **NULL/DUMMY** objects, named as follows:

SUSP_LF
SUSP_LR
SUSP_RF
SUSP_RR

The NULL/DUMMY must be designed to move the suspension on the **Y** axis.

Below an example of how to animate a suspension correctly:



NOTE: Before exporting the FBX, timeline needs to be set with the same number of total frames as the number of animated frames created.
Example: If you animate 20 frames, do not export with a timeline of 30. This will cause a crash. Remember to set the timeline to 20 if you have animated 20 frames. Empty frames will cause a crash and are not supported.

Suspension Hierarchy

The suspension must have the hierarchy identical to one of the two FBX examples provided:

TEMPLATE_Suspension_EASY.fbx and **TEMPLATE_Suspension_COMPLEX.fbx**

The first scene contains a simple suspension hierarchy, made for a car with simple suspension system.
The second is prepared for complex suspension hierarchy, like 60's Formula 1 cars, with more complex arms and particular suspensions. Those examples contains more **DUMMY/NULL**s

Some names can be customized and we have named the customizable **DUMMY/NULL** in an appropriate way, inside the template.fbx
"**Custom_name##**". (where # is a number)
All the **DUMMY/NULL** are used for animated parts. Their use is optional. You can create the necessary number of **DUMMY/NULL** as you desire.

The following **DUMMY/NULLs** are **mandatory**:

Suspension Null's:

SUSP_LF	Left Front
SUSP_LR	Left Rear
SUSP_RF	Right Front
SUSP_RR	Right Rear

Hub Null's:

HUB_LF	Left Front
HUB_LR	Left Rear
HUB_RF	Right Front
HUB_RR	Right Rear

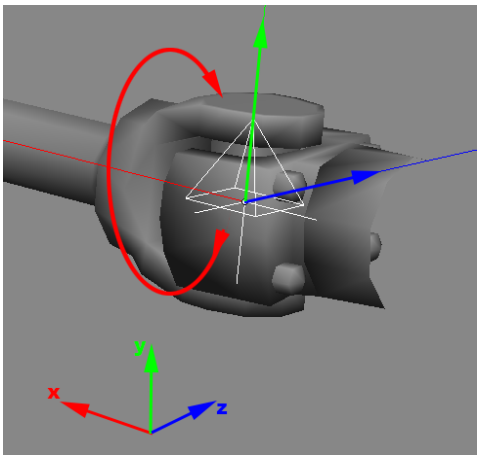
Wheels Null's:

WHEEL_LF	with linked the TYRE_LF for the tyre mesh, RIM_LF for the Rim mesh, and RIM_BLUR_LF for the Rim Blurred mesh
WHEEL_LR	with linked the TYRE_LR for the tyre mesh, RIM_LR for the Rim mesh, and RIM_BLUR_LR for the Rim Blurred mesh
WHEEL_LR	with linked the TYRE_LR for the tyre mesh, RIM_LR for the Rim mesh, and RIM_BLUR_LR for the Rim Blurred mesh
WHEEL_RR	with linked the TYRE_RR for the tyre mesh, RIM_RR for the Rim mesh, and RIM_BLUR_RR for the Rim Blurred mesh

In some cars, the transmission shafts might be visible. There is a convention name to animate automatically these parts.
Trasmission **DUMMY/NULL** names:

TRANSMISSION_L_1 for the Left shaft
TRANSMISSION_R_1 for the Right shaft

If you have more transmission pieces to animate, you can use sequential of numbering. For example: TRANSMISSION_L_2 , TRANSMISSION_L_3 and so on. Same thing for TRANSMISSION_R_2 and so on.
The engine recognizes the prefix "TRANSMISSION_L_" and searches for a sequential number after. There is no hardcoded limit on the pieces of trasmission that can be animated.



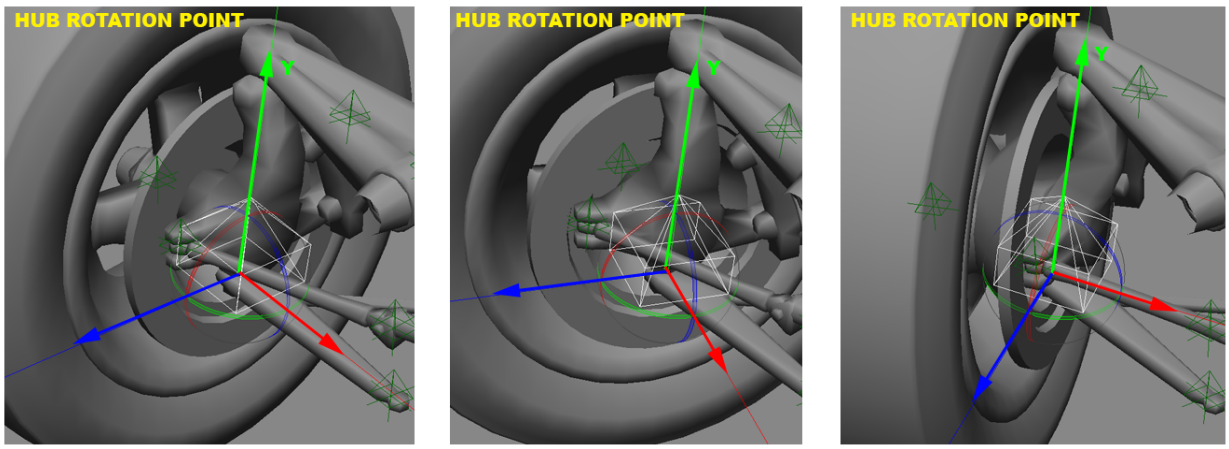
The tramission nulls rotate on the **X** axis and needs to be oriented like the image above:

This node is useful for animating the gimbal on **Y** axis according to the suspension animation, and the engine automatically rotates the transmission according to the wheel on **X** axis. Avoid to animate on **Z** axis this Null, as it is not used.

NOTE: In order to have a correct direction of rotation, the Z axis of the transmission nulls always needs to head forward.

The engine recognizes other 4 nodes for the the hubs of every wheel. These nodes are made to allow a rotation of the hub, related to the camber of the wheel.

HUB_LF
HUB_LR
HUB_RF
HUB_RR



In the above image example , the hub is father of the steer arms.
So you can animate up and down movements of the hub or the suspension and during the animation you can change the camber of the hub as requested by the physical suspension scheme.
Inside the **TEMPLATE_Suspension_COMPLEX.fbx** you can find an example of the correct hierarchy.

Note: The **SUSP_** node must always match the position of the **Wheel_** node. The AC engine, verifies the position of the suspension in 3D space by checking the **SUSP_** node. The wheel bounding box is recognized between the **SUSP_** node and **Wheel_** node. Those 2 positions must be the same.
Again the FBX file **TEMPLATE_Suspension_COMPLEX.fbx** is a perfect example .

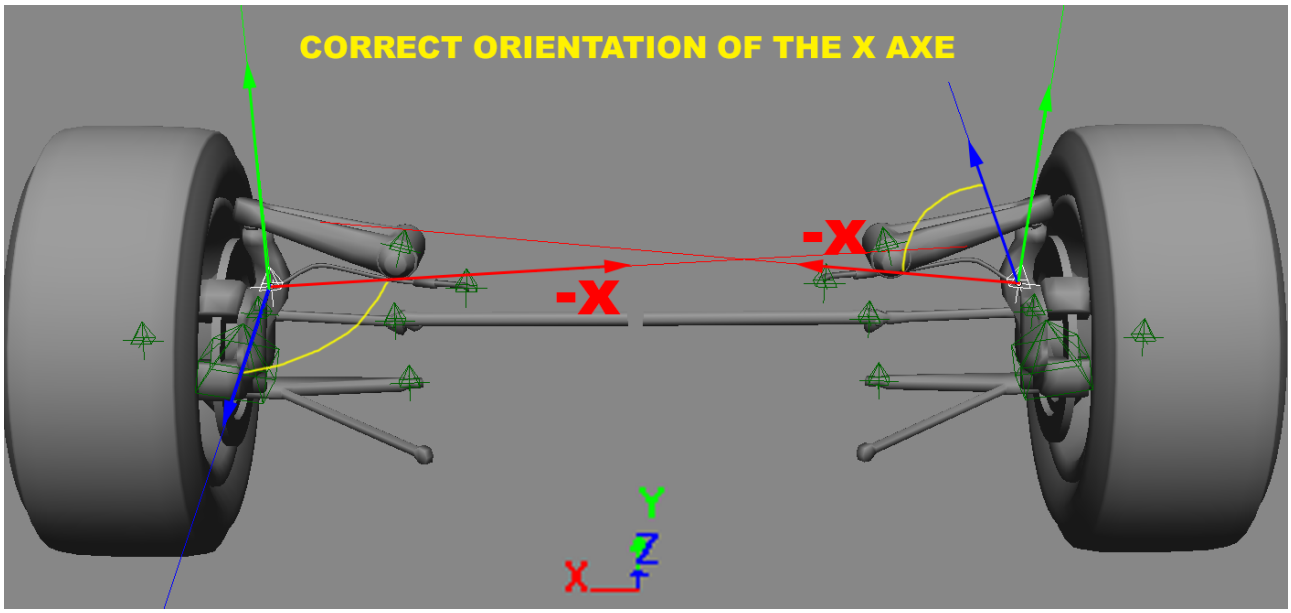
STEER ARMS and DIRECTION COSTRAINT

We can animate many different parts and just import the animation to the editor and from that export to the game, but the **STEER** arm **cannot be animated**. Its position changes in the 3D space according to the HUB rotation.

In order to constrain the movement of the steer arm to the HUB's position and rotation, the convention name with a prefix "**DIR_customName**" must be used. This indicates the direction of the **X** negative axis of this mesh, and the mesh called "**customName**" will head always the X axis on the exact direction.

Example: A null called "**SteerArm_L**" will point the negative X axis in direction of a null named "**DIR_SteerArm_L**"

Pay attention to the rotation of the null that has attached the animated mesh. In the image below the right Null point has Z axis positive. The left Null point has Z axis negative. This allows the -X axis to point to the center of the car or any other direction needed by the mesh.



Inside the **TEMPLATE_Suspension_COMPLEX.fbx** file, you can find a proper hierarchy example.

Note: You can create more costraints, if you have more objects to costraint to the HUB by simply giving different names. Nevertheless, it is always good in terms of optimization to use the smallest possible of constraints.

You can animate your custom null's in the following vectors: **Rotation** - **Translation** - **Scale**.

Inside the **TEMPLATE_Suspension_COMPLEX.fbx** example file, you can see the animation of the suspension spring, on **SCALE Y** .
Note: **Never animate the mesh.** Always animate the NULL's only. With this approach you can change and update your mesh everytime you want without re-exporting the animations.

Use the same technique to create animations for any **NULL/DUMMY** that must be animated. For example, doors, gearbox levers, or any other part.

ANIMATION EXPORT

Once you have animated your **NULL/DUMMY** inside the editor car scene, you need to export the animation from the editor to AC specific **clip (**file format), called **name.ksanim.**

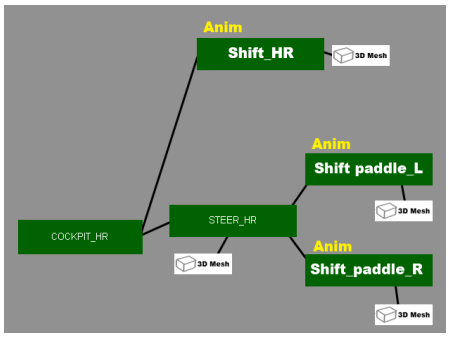
A couple of rules must be followed:
1) To generate a **clip**, from the editor, you need only the animated nulls. The mesh is not needed. You can also export a mesh, but the editor exports only the objects that have animation keyframes. As we said above, it is a good technique to animate only the NULL/DUMMY so that you can change the animation independently from the actual 3D mesh.

2) When you export an animated null, you must also export the the hierarchy tree above it, because the name of an object inside the editor, is determined from its position in the hierarchy tree.

3) Always export using the FBX format, as is the only one that supports animation. Do not use any other formats.

In the example shown in the image at the left, we have 2 hierarchies of nulls, that contain meshes as children. We animated the shift and steer paddle and we need to export the animation.

We can't export **SHIFT PADDLE_L** only. We must take the hierarchy from **COCKPIT_HR**, including **STEER_HR** and **SHIFT paddle_L** .
This way the editor will define a node related to the position of **SHIFT PADDLE_L** in the hierarchy.



Animation optimizations.

1) The frames are interpolated in the game. You don't need to export an animation with all the keyframes. For simple animations, like doors, gear levers and so on, you can export just the important frames only.
For a simple door animation, the “close” and “open” frames are enough, the game will interpolate the rest. When you have more complex doors, with pistons, vertical openings, like those in a Mclaren P1, you can add more frames to animate the door in a more precise way. But always keep in mind that the less frame you use, the more optimized the result will be, because the engine interpolates smoothly between keyframes.

2) Identical frames are optimized. If you create multiple identical frames, and the variation between them is 0, then the frames will be optimized.
Example: A gear lever starts animation at frame 15 of the complete animation, because during previous frames it stays fixed to its position, waiting for the reach of the hand of the driver. A keyframe must be placed to frame 0 and another one at frame 14 both in static position. On frame 15 starts the animation of the gear lever.There is no need to place more keyframes from 0 to 14.

CLIP NAMES AND CONVENTIONS

Two types of pre-programmed playback:

1) PingPong The animation played reaches the end is then played in reverse to the start.
2) Loop when the first frame match the last frame and the animation restarts the loop.

Specific convention names have pre-programmed playback in game, We manage the playback automatically using LOOP or PINGPONG, so you can prepare your clips with this logic.

DRIVER ANIMATIONS CLIP NAMES

steer.ksanim	**Loop** for the rotation of the driver's arms on the steering wheel
shift.ksanim	**PingPong** for the animation of the driver's arm to the gearshift lever (we usually do a simple animation, check the fbx example)
shift_dw.ksanim	**PingPong** for the animation of the fingers that operate the paddle shifts down (usually left)
shift_up.ksanim	**PingPong** for the animation of the fingers that operate the paddle shifts up (usually right)
claim.ksanim	**PingPong** to claim other drivers (moving fists in the air etc.)
celebrate.ksanim	**PingPong** Victory gesture

CAR ANIMATION CLIP NAMES

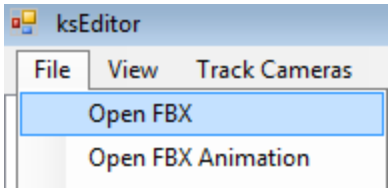
car_shift.ksanim **PingPong** to animate the car gear lever
Important: this clip must have the same number of frames of shift.ksanim from the driver to match perfectly the arm movement with the shift animation

car_susp_LF.ksanim	**Controlled by engine** for the animation of the Left Front suspension
car_susp_LR.ksanim	**Controlled by engine** for the animation of the Left Rear suspension
car_susp_RF.ksanim	**Controlled by engine** for the animation of the Right Front suspension
car_susp_RR.ksanim	**Controlled by engine** for the animation of the Right Rear suspension
car_door_R.ksanim	**PingPong** for the animation of the right door opening (the close animation should be the open animation in reverse. Do not animate the closure too.)
car_door_L.ksanim	**PingPong** for the animation of the left door opening (the close animation should be the open animation in reverse. Do not animate the closure too.)
car_wiper.ksanim	**Loop** for the animation of the wiper (here you must animate the full animation back and forth)
lights.ksanim	**PingPong** for the animation of the car lights that are mobile (i.e. Ferrari F40. Animate only aperture, then play reverse for closure.)
car_shift_up.ksanim	**PingPong** for the animation paddle shift up
car_shift_dw.ksanim	**PingPong** for the animation paddle shift down

Note: You can create all the animations needed. You can also use new names and then call them from an .ini script. But the names listed are recognized automatically from the game and managed properly.
For example: the animation **car_wing.ksnim** is a custom name. On certain cars we created animations called **Wing_Rear.ksanim** or **Wing_side.ksanim**. All of them are managed from .ini scripts.

car_wing.ksanim	**PingPong** for the animation of movable wings (animate only one part of the movement then play in reverse)

EXPORT ANIMATIONS FROM THE EDITOR



Once the car and its animations are created in the editor, it needs to be **saved** as a .kn5 file, that can be readed from the game.
The rules to export a car in the game are the following:

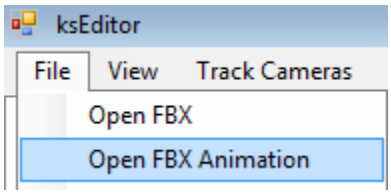
1) All the meshes must have one only UV coordinates set;
2) The Null/Dummy essentials must be present in the scene;
3) All the materials must have a texture assigned to their texture slot. (and we suggest to use always DDS and not png as they are not compressed).

In order to **EXPORT** a car, the following nodes (at least) are required:

SUSP_LF
SUSP_LR
SUSP_RF
SUSP_RR
WHEEL_LF
WHEEL_LR
WHEEL_RF
WHEEL_RR
RIM_LF
RIM_LR
RIM_RF
RIM_RR
RIM_BLUR_LF
RIM_BLUR_LR
RIM_BLUR_RF
RIM_BLUR_RR

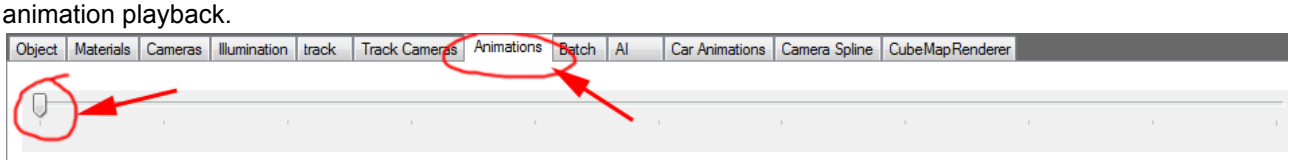
without one of these nodes, AC will crash (check the template example).
There are many other nodes used from the engine, but marked as optional since, even if not present, the game will work anyway. The above mentioned required nodes needs to be present on **LOD_B** and **LOD_C** too.

To export animations, follow these steps :
- Open the FBX car in the editor.



- Load the FBX animation previously exported in the “animations” folder.
When an animation is loaded it automatically saves a **nameclip.ksanim** file in the same folder of your fbx.

- At the bottom part of the editor UI select the Animation tab. Drag the animation slider , and you can see the animation playback.



You can load another animation and so on. Every time you load an animation, a **.ksanim** file is saved with the same name of your source FBX and the .ksanim extension.
So you have to rename your clips to match our name conventions like for example **steer.ksanim** for the arms of the driver, and so on.

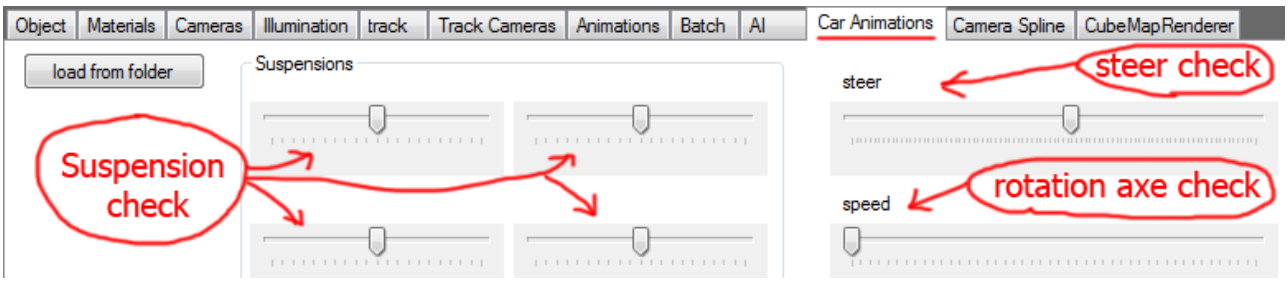
CHECKING CAR ANIMATIONS

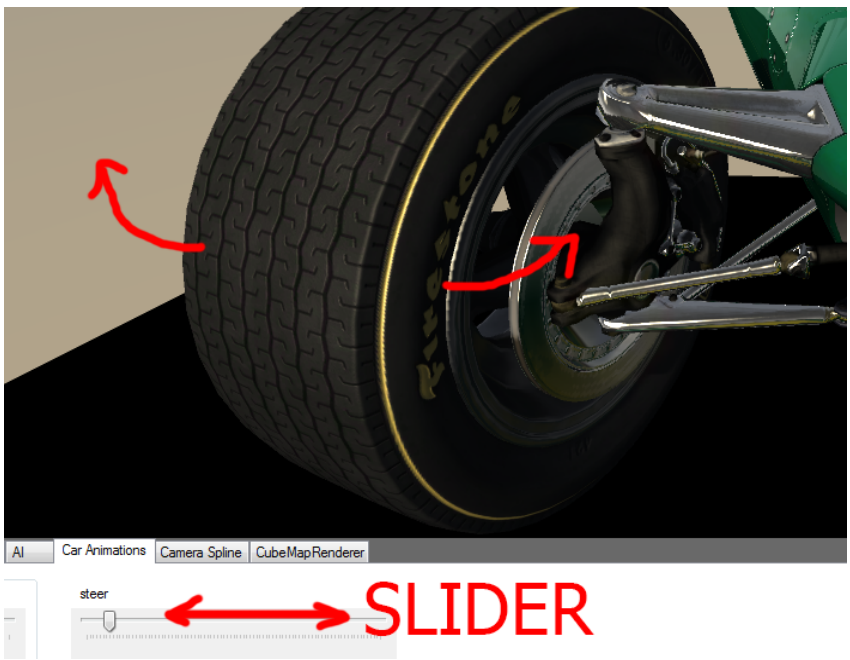
When you have exported all your suspensions **clips** in the proper animation folder (see “Asset Organization” section) You can load your car.fbx in the editor and check if the animations work properly.
You can do this only after you have create the animated **clips**.

NOTE: **LOD_B** must contain the same null hierarchy and names of the animated nulls from **LOD_A**.

To check do this: Open you car.fbx

After the car is loaded, you can load the clip of the suspension that you want to test.
In the tab called “Car Animations” you’ll find some sliders. These are made to test the spring animation, the costraint of the arms and the wheel rotation center.

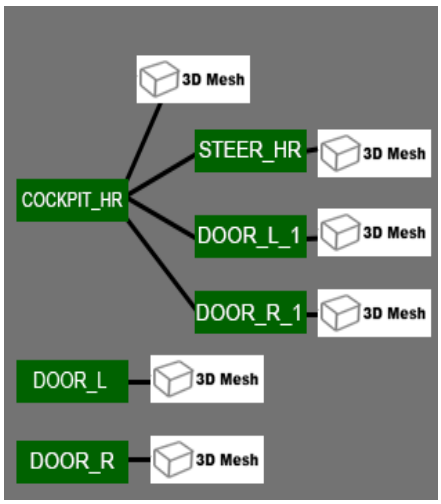




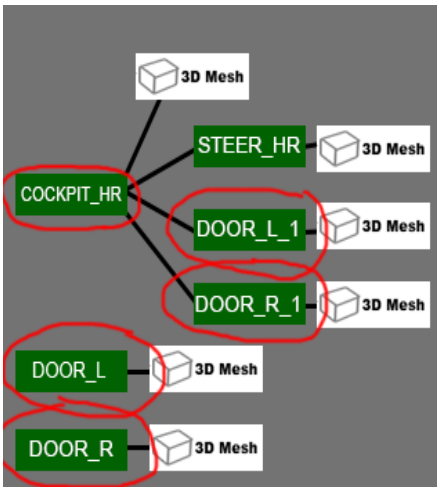
The sliders allow you to check the suspension work in the editor and detect any issues, frame by frame. Here you can see an example: Moving the slider you can check the hub behaviour.

GENERIC ANIMATION EXPORTING TRICKS

There are some important things to keep in mind regarding animation exporting, especially about door animation clips. When exporting a door clip, the following hierarchy should be present:



Note that the meshes of the cockpit doors are under the null, **COCKPIT_HR**. This is needed because the cockpit switches from high to low resolution. The LR door will be hidden. So we have duplicated the door animation nulls with animation included, and placed them outside of **COCKPIT_HR**. When you export, remember to include all the cockpit door nulls.

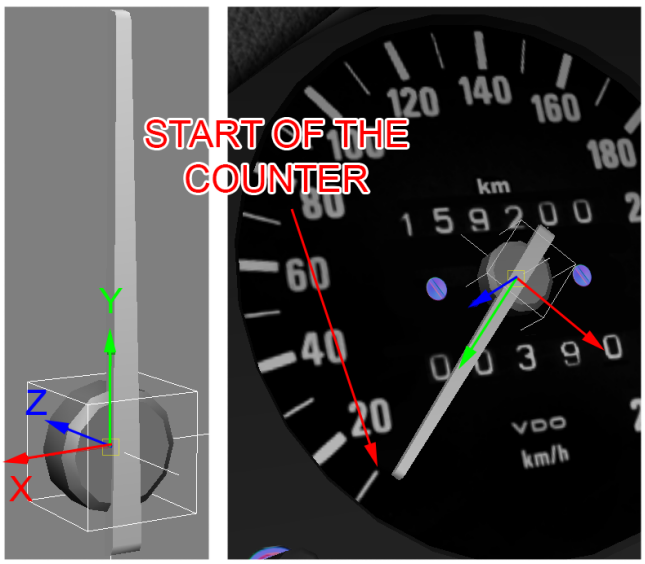


DASHBOARD NEEDLE

To animate a needle on the dashboard, an **ARROW_** mesh needs to be placed under a proper null. Like other objects, **ARROW_** meshes follow specific conventions.

Name conventions :

ARROW_SPEED	for the speed indicator
ARROW_RPM	for the rpm indicator
ARROW_TURBO	for the turbo pressure
ARROW_FUEL	for the fuel quantity



Each indicator created, must be managed by specific values in the **analog_instruments.ini** script, inside “content/cars/car_name/data” folder.

The **ARROW_** mesh must be created in neutral position and linked as child of a specific null.
The arrow must be oriented like in the left image:
The Y axis determinates where the arrow position is on the counter, and must be placed on the ZERO or the start of any indicator.
Note: Z axis must always heads forward.

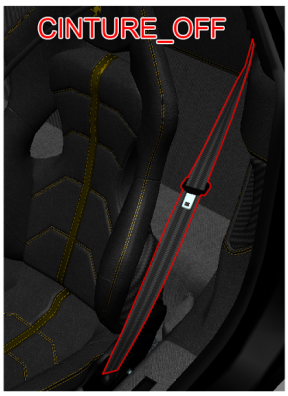
SEAT BELTS

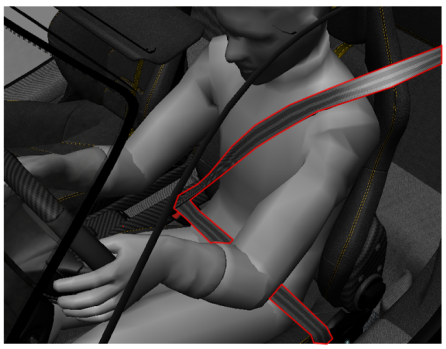
The cockpit contains two different meshes of the belts: One for the belt on and another for the belt off. These two meshes must be linked as a child of the null **COCKPIT_HR** and must be named as follows:

CINTURE_ON for the belt on the driver when is driving
CINTURE_OFF for the belt on the seat, without driver

These two meshes are the only in the pipeline that need specific name on the mesh.
NOTE: We apologize for the two names in “ITALIAN” language (CINTURE = BELTS) , but it’s too late to change it. So… the names for belts are in italian….

To create the proper mesh of the belt on a driver, place the driver first, then animate him and verify how the arms move in order to avoid compenetration with the belt mesh.

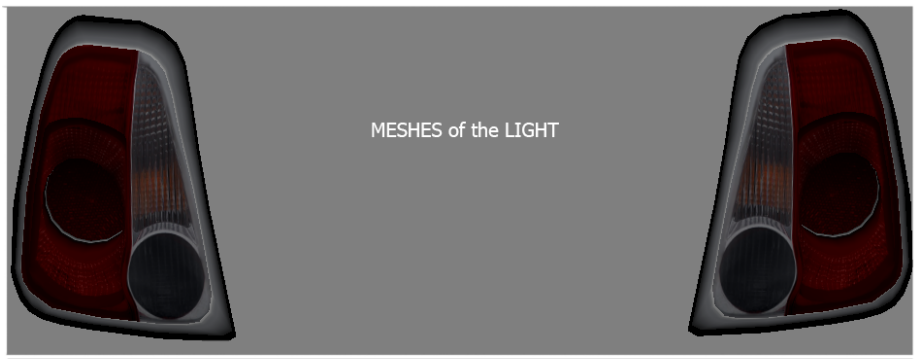


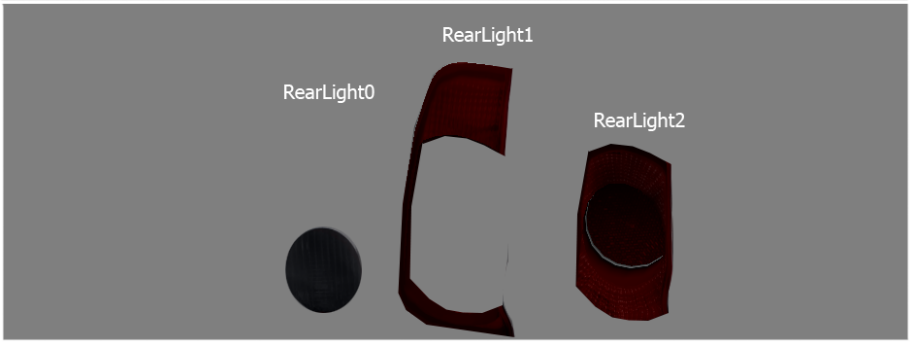


CAR LIGHT MESHES and SCRIPTS

Each car must have individual meshes for light ON and OFF conditions. The light meshes must be separated and detached from the chassis of the car. The mesh name must be pointed by the **lights.ini** script. The same scripts includes the instructions for the ON/OFF conditions, as well as the light emission colour.

Example image on the right: The mesh of the light is made from different pieces, named at your discretion. They are divided according to their function. Some examples:
Position lights, brake, rear, standard front lights, high beams, and so on.
You can split your mesh as you prefer.





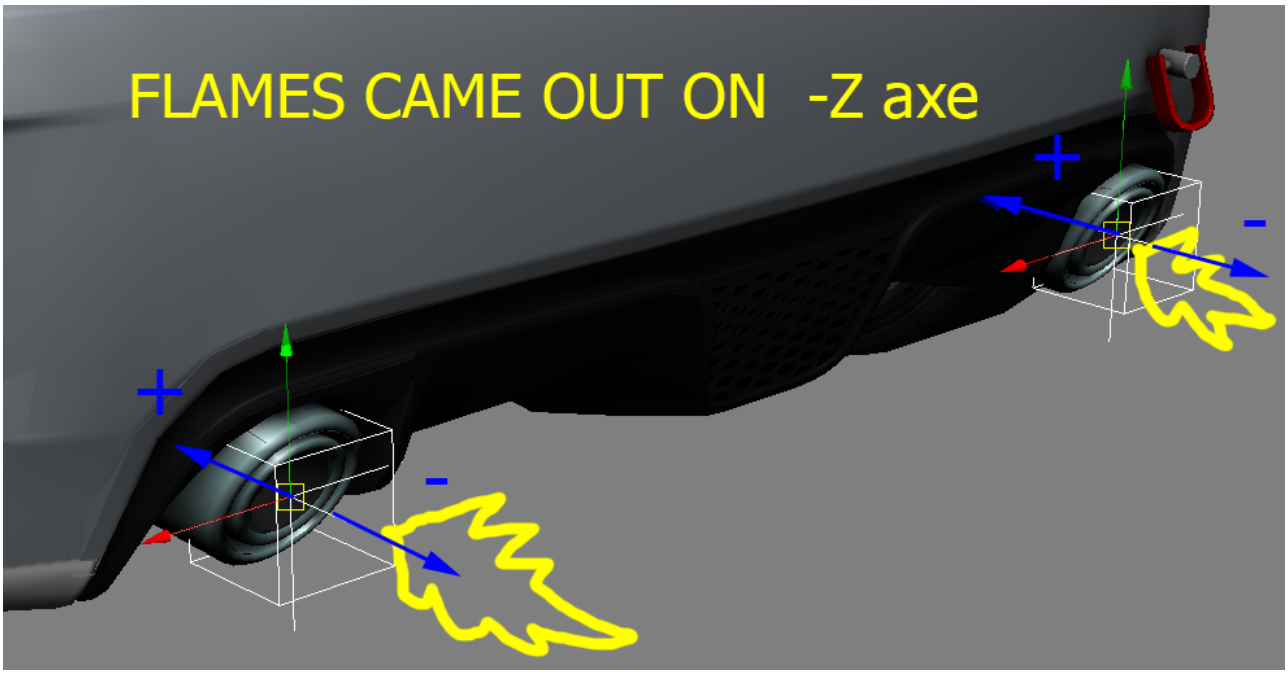
Note: There is no need to split the lights in “right” and “left”. They can be one mesh because they turn on together.
Open the **lights.ini** script located in the “data” folder
Inside, the script contains the following values:

```
[BRAKE_0]
NAME=RearLight2
COLOR=25,0,0

[LIGHT_0]
NAME=LightFront1
COLOR=84,100,300

[LIGHT_1]
NAME=LightFront2
COLOR=2,2.3,3.5

[LIGHT_2]
NAME=LightFront3
COLOR=3,0,0

[LIGHT_ANIMATION_0]
FILE=lights.ksanim
TIME=0.3
```

In the above example: The **NAME=** value assigns a mesh called **RearLight2** (as seen on the image above)
The **COLOR=** value assigns a colour when the car brakes.
Different functions and colours can be assigned to different meshes. As an example, the value **COLOR=3,0,0** assigns a specific colour to the light. Meshes are lighted with HDR and do not have a max limit of intensity.
The values for the **COLOR=** parameter are in RGB **0 to 1** range, so a value of 1 means the maximum value of the RGB scale (256). Values can go over 1 if more intensity is needed. As an example, a value of 300 is given to the **[LIGHT_0]** section, in order to produce a strong HDR intensity glow.

CAR EXHAUSTS

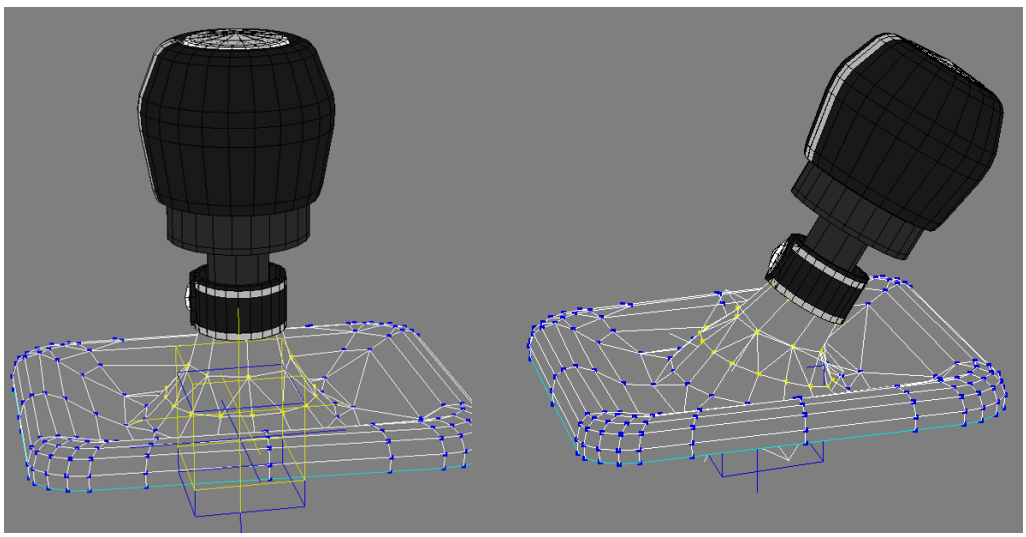
In order to define the backfire flame position, place a null named **EXHAUST#** where **#** is a number from 0 until the necessary frames of the flame animation.
For example: **EXHAUST1**, **EXHAUST2**, and so on.
The flames spawn from the rear of the exhausts, and follow the Z negative axis.



SKINNED MESH

FBX skinned mesh is supported. **Skinned mesh** can have as many bones as necessary, but no more than 4 bones influencing a single vertex. A good example of skinned mesh is the driver (explained later) or the gearshift lever with a fabric coat at the base of the lever.

The example image at the right, showcases the use of bones.



Rules for skinned mesh:

1) All vertexes must be influenced by at least one bone. If a vertex is not influenced by a bone, its world coordinates will be 0,0,0 resulting in a long polygon that spawns from the center of the 3D world.

A non skinned object can be linked with a skinned mesh. Connect the non skinned object as a child to the skinned mesh. In the above image, the skin has 2 bones but the handle is a parent of the yellow null without any skinning.

2) Every material with a skin rig must be unique.A material cannot be used on a standard mesh and at the same time on a skinned mesh. Two different materials must be created, one for the standard mesh and another one for the skinned mesh.

3) The skinned dedicated material, must be named as KsSkinnedMesh
The animation works only when the **KsSkinnedMesh** material is assigned properly to the skinned mesh.

NOTE: Do not use this material on a standard mesh.

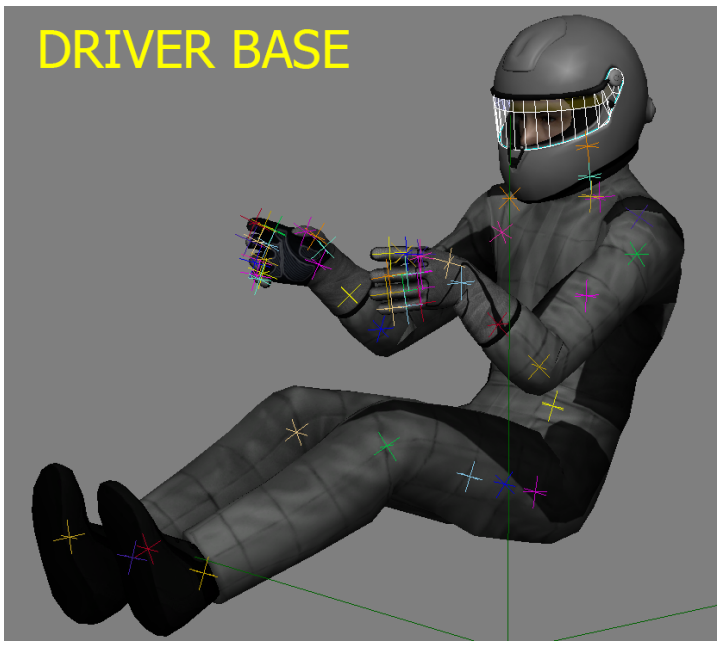
DRIVER POSITION AND OBJECTS

A copy of AC driver with the bones skinned, basic animation of the steer rotation, helmet and some textures, is provided as an example template.It can be placed inside any custom car.

For a proper placement follow these steps:

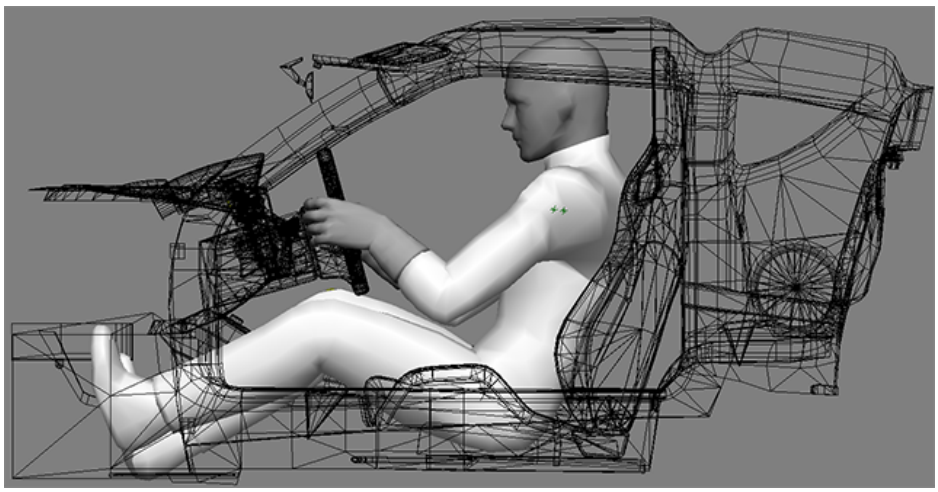
1) If you want to use a custom driver mesh go to the point CUSTOM DRIVER, else go to the next step.

1) Import the template file **DRIVER_BASE.fbx** in your 3D application. You should see the driver as in picture.
Inside the template, a basic steering wheel rotation animation is provided as an example.
The animation is made of 200 frames.
The neutral position is on frame 100. From neutral (100) to 0, the steering wheel turns left. From neutral (100) to 200, it turns right.



2) place your driver on the seat, with his hands on the steering wheel. Probably some modifications of our animation template will be necessary.

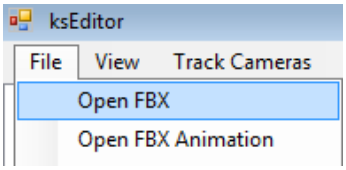
The image below shows an example placement:



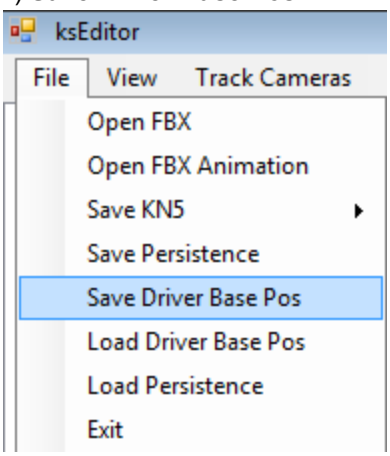
Driver mesh and position can now be exported and it will contain the exact hierarchy provided, and the correct names of the bones and various objects.

How to export the driver base position from the editor:

1) Open in the editor the FBX file with the driver placed in the correct base position.



2) **Save Driver Base Pos**



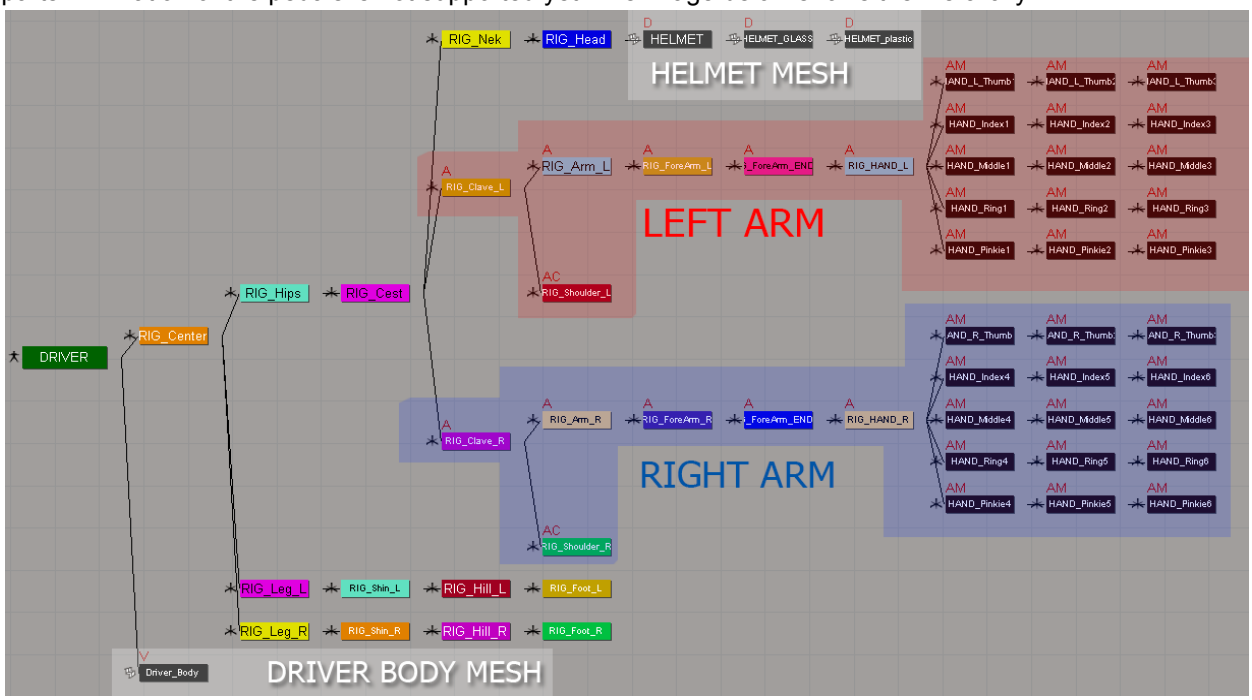
A file named **driver_base_pos.knh** is created and stored in the same folder where the FBX was loaded.
This file must be placed in the following path: **AssettoCorsa/content/cars/CAR-NAME/** where car-name is the car folder.
The game engine will load the driver and place it the correct position, stored in the **driver_base_pos.knh** file

DRIVER ANIMATIONS

The provided template file **DRIVER_BASE.fbx** contains a basic example of 360° steer rotation loop animation. This animation will probably not match the steering wheel of your design. The animation must be modified to match your custom steering wheel dimensions and placement.

Note: The animation must be **200** frames, and frame **0**, frame **100** and frame **200** must match, to allow the animation **LOOP**.

After editing the animation, save the keyframes of only the arms NULLs and export the FBX just the animated parts. Animation of the pedals is not supported yet. The image below shows the hierarchy:



The bones of the arms are highlighted in the blue and red area in the image, and every bone is parent of the **RIG_Clave_L** and **RIG_Clave_R** bones.

To animate gear change hand, animate the arm bones from **RIG_Clave_L/R** to the fingers.
To animate paddle gear change, animate the fingers only.
For every animation you must export a copy of the **driver.fbx** with only animated the part needed for the desired cllip. Example: Export **driver.fbx** with the steering wheel animation only, then another one with gear animation only, and so on.
Store the driver animations with the names indicated below in the related animation folder of the car fbx datas.

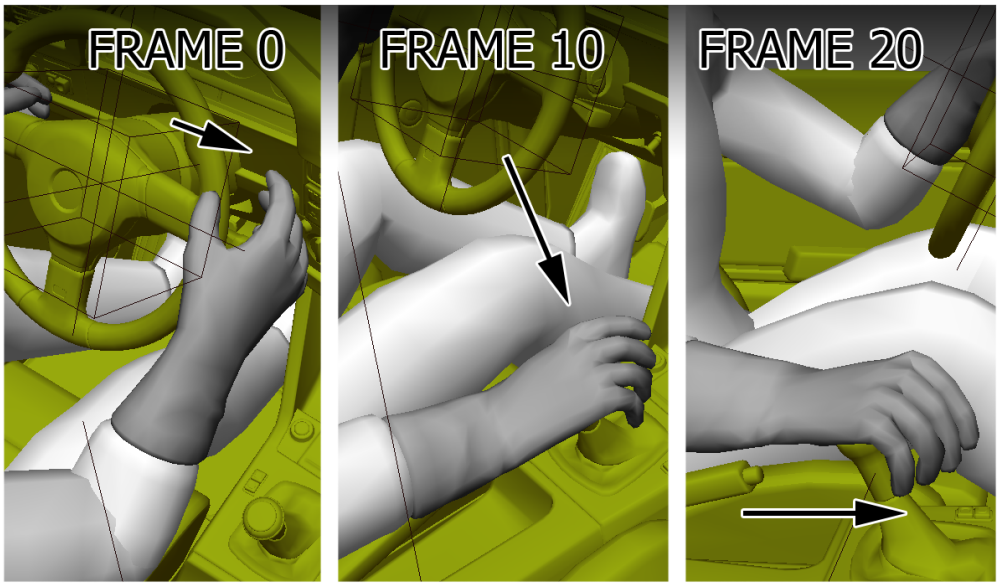
Steer.fbx for the 360° steer rotation
Shift.fbx for the gershift animation
Shift_up.fbx for the paddle shift up
Shift_dw.fbx for the paddle shift down
claim.fbx for the claim gesture
celebrate.fbx for the victory gesture

Check chapter **EXPORT ANIMATIONS FROM THE EDITOR** for instructions on how to create a clip.

Note: Always verify that the car shift animation and the driver shift animation have the same number of frames so that they will animate synchronized, while in game.

Warning: There is typo in the name of the "neck" bone, which is typed "nek" by error. Although wrong, the game still works with this wrong name, so please do NOT correct the typo and keep it "nek".

Example: When the driver changes gear, his arm starts the animation with the hand slightly distant from the steering wheel. (see image below)



On frame 0 the hand is slightly away from the steering wheel. On frame 10 the hand is on the gear lever. On frame 20 the hand moves the gear lever. Be sure that the gear lever animation is synchronized with the hand. For example, if the hand needs 10 frames to reach the gear level, the gear lever must have 10 frames stopped before it starts its movement.

DRIVER SCRIPTS

The driver is managed by **driver3d.ini** script in "**AssettoCorsa/content/cars/CAR-NAME/data**".
The file structure is the following:

```
[MODEL]
NAME=driver
POSITION=0,0,0
```
- This section determines the model of the driver to use (there are different models available)

```
[STEER_ANIMATION]
NAME=steer.ksanim
LOCK=360
```
- This section determines the clip to use for the steering wheel animation and its rotation limit (in this case 360 degrees)

```
[SHIFT_ANIMATION]
BLEND_TIME=200          ; (MS) Time used to move the driver's hand
                        from the steer position to the first frame of the
                        animation.
POSITIVE_TIME=400       ; (MS) Time needed to move the driver's hand from the
                        first frame of the animation to the gear lever (forward
                        animation).
STATIC_TIME=10          ; (MS) Interval of time were the driver hand is still on
                        the gear lever (Wait Time between forward and reverse
                        animation)
NEGATIVE_TIME=400       ; (MS) Time needed to move the driver's hand from the
```

```
                        gear lever back to the first frame of the animation
                        (reverse animation).
PRELOAD_RPM=6000        ; (MS) when the engine reaches this RPM value the
forward
                        animation is automatically played

INVERT_SHIFTING_HANDS=0 ; Set to 1 if the driver upshifts with the
                        left hand.

[HIDE_OBJECT_0]
NAME=DRIVER:HELMET            ; Hide the specific mesh (copy the correct name
from the
                        editor) when in cockpit camera. In this case, the helmet
                        is hidden...

[HIDE_OBJECT_1]
NAME=DRIVER:Driver_Body_SUB1  ; … and here the driver's head mesh is hidden.
```

Note: The driver face mesh can have different name if you use a custom driver mesh.

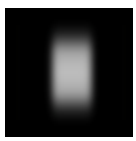
CAR SHADOWS

The car ground shadows are not generated in real time, like the sun shadows, but they are very important in order to improve the visual effect of the ground position of the car and emulate an ambient occlusion on the ground.
There are five shadow textures for each car. Four dedicated textures for each wheel and another one for the car body.
If not present, the car body texture is automatically generated once in game, otherwise an existing one is used.
A pre-made texture is used for the wheels shadows. All shadows must be placed in the root custom car folder (see "Asset Organization")

Examples:

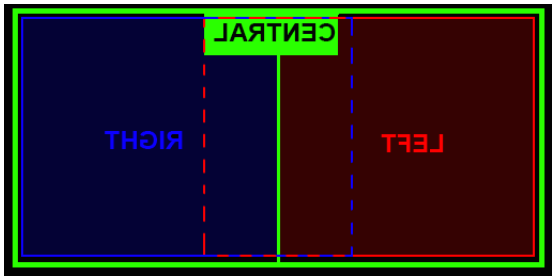
This is the BODY shadow of the FIAT 500 car. In game it looks as in the right image example: The shadow is blended with the sun one.

Remember to place in the folder also the 4 wheel textures (see example on the right image). Those work in the same way as the body shadow, but they are attached to the wheels. You can take the automatically generated car body shadow and further refine it in photoshop or your favourite application.



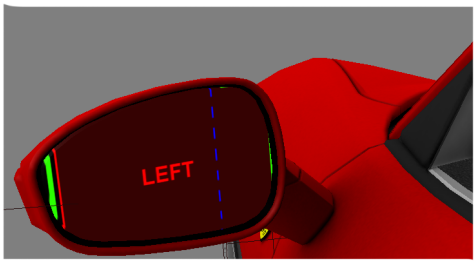
CAR MIRRORS

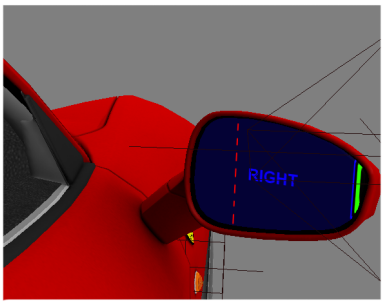
In order to make car mirrors work, a material must be created (the name is not important), and assigned to the polygons of the mirrors. The mesh must be mapped with the provided texture called **MIRROR_PLACEMENT.PNG**

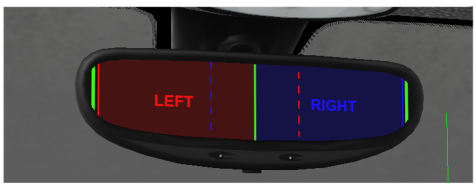


This texture is mirrored and must be mapped on the mirror mesh in order to visualize it correctly. The texture is divided various areas. The green area is what is visible from the internal central mirror, the red from left hand mirror and blue from right hand.

Note: always remember to keep correct aspect ratio of your UV on the mirrors polygons, or the image of the reflection will be streched.



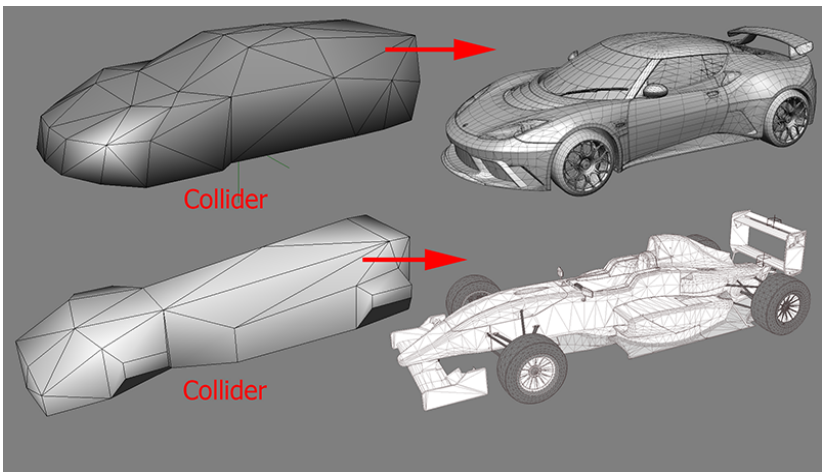




Examples of the **MIRROR_PLACEMENT.PNG** file, applied to the various mirror meshes

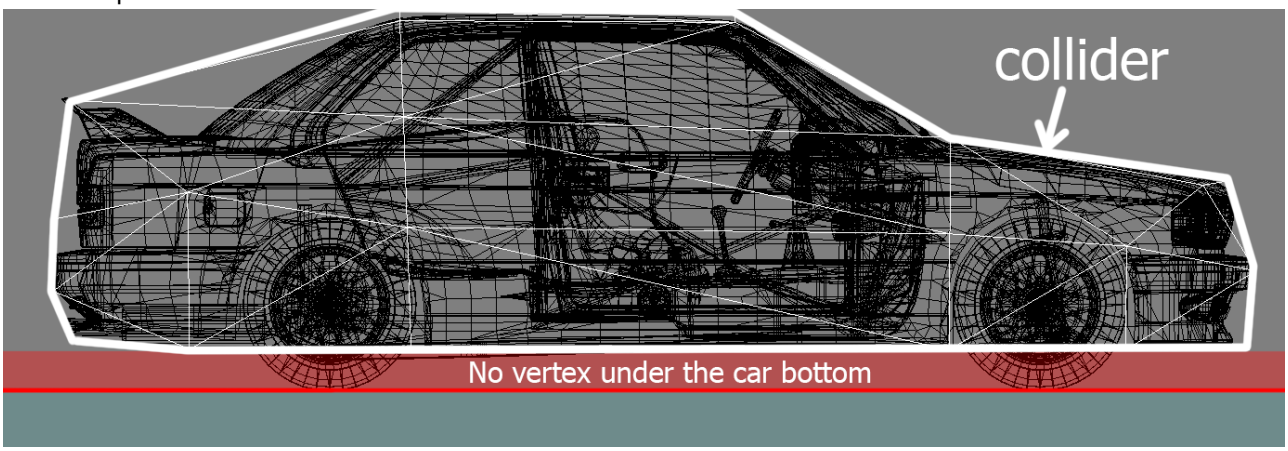
CAR SIMPLE COLLIDER

Collisions between vehicles are one of the most resource demanding activities of the game, especially if 20 cars collide in a corner all at the same. To optimize such situations, a simple collider shape is used to calculate collisions between car bodies and tracks objects.
The collider shape must be a simple solid with as low polygon count as possible, roughly 40 or 60 triangles, without UV and without material.



Rules for collider objects:

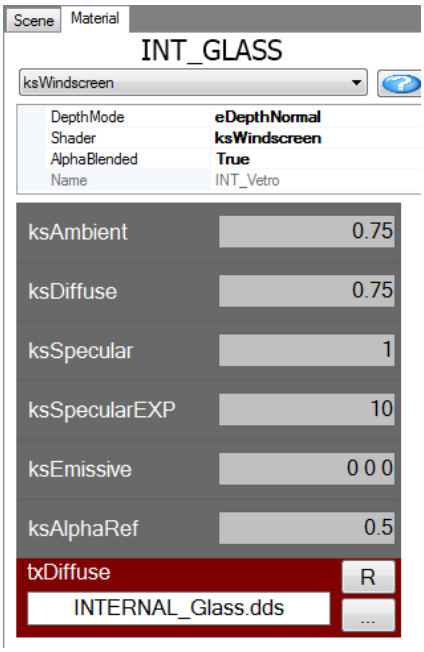
1) The collider should have no more than 40/60 triangles.
2) A material called "**GL**" must be assigned to the collider inside the editor. This is a special material specifically made for meshes that are not rendered. Meshes with this material are used only for collisions.
3) The collider must not extend with any vertex, under the floor of the car.
See example:



4) The collider must have **no holes**. The mesh must be **closed.**

Once the collision mesh is done, simply save the fbx from the editor like a kn5 file, using name "**collider.kn5**". The file must be placed in the same folder of the car LODs.

INSIDE WINDSCREEN AND DOORS GLASS

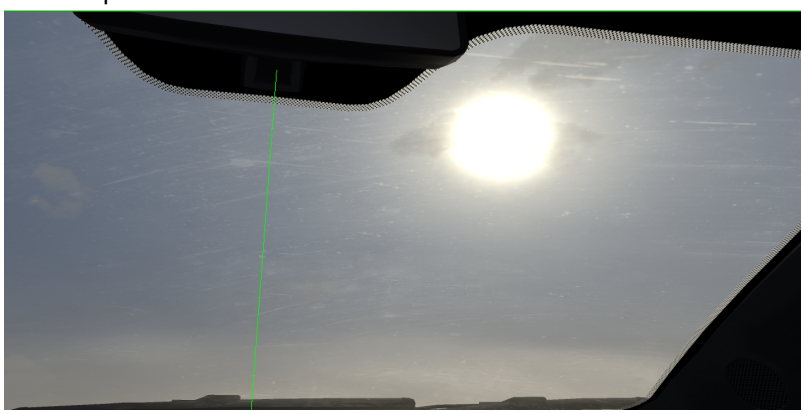


A different shader is used for the internal of the cars, made specifically to emulate the sun reflection effect when the surface is not totally clear. This effect can be increased or reduced by editing some glass texture parameters.

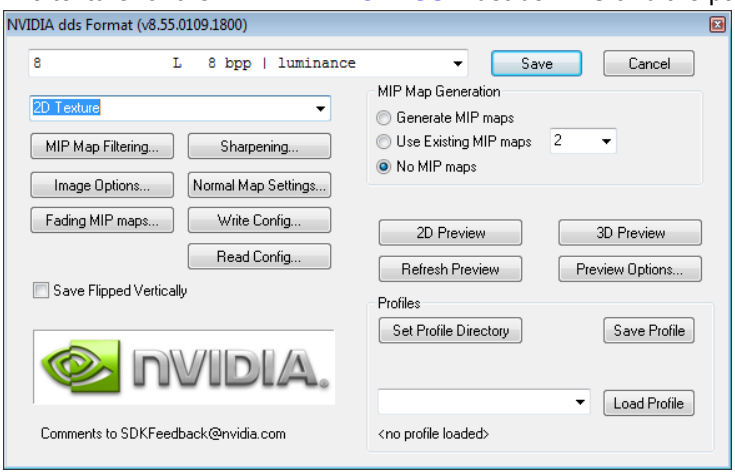
The shader to apply at the internal glass is **KsWindscreen**

NOTE: Glass meshes **must** be parent of the **COCKPIT_HR** null and doors internal glass must be parent of the doors nulls under the **COCKPIT_HR**, This is needed because internal glass switches together with the cockpit_HR and cockpit_LR LODs)

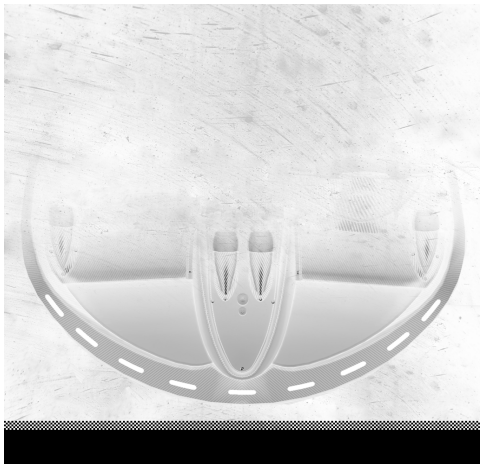
Shader parameters are shown in the left image.
An example of the shader effect is shown below.



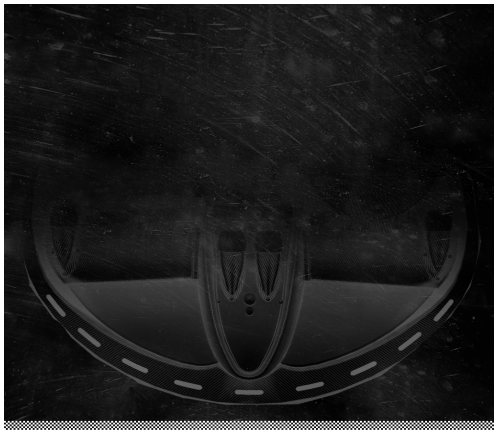
The texture for the **INTERNAL GLASS** must be DDS and the parameters as in the following image:



The texture must have an alpha channel and the following configuration



On the **left image** is the diffuse of the glass texture.
On the **right image** the alpha channel.
A soft shadow of the cockpit dashboard is painted on top of the texture. This trick allows an emulation of the internal reflection of the dashboard on the glass when the sun is forward.



Note: The internal glass mesh is just a copy ot the external polygons of the glass, but the internal glass should all the normals pointing internally. Do likewise for al the windows internal glasses.

EDITOR “CAR SPECIFIC SHORTCUT KEYS”

Specific shortcut keys have been added to the editor that allow faster work and quick feature testing.

Shortcut keys
F1 Switches from RIM to RIM Blurred. The effect becomes visible and can be tuned faster.

F3 Switches from **COCKPIT_HR** to **COCKPIT_LR** (if exist) and vicecersa. Usefull to see if the switch of the cockpit models is too abrupt.

F4 Shows the **GLASS** damage mesh on and off.
Note: This command works only if the glass damage mesh NULL exists.

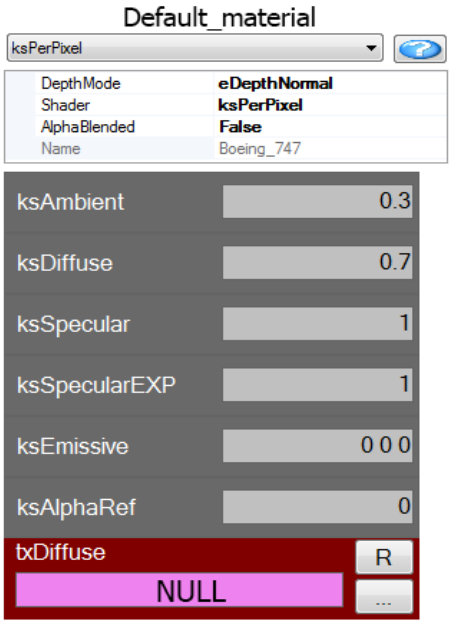
MATERIAL EDITOR SHADERS AND PERSISTENCE FILES

Warning: This section is still work in progress, features missing or do not work perfectly.

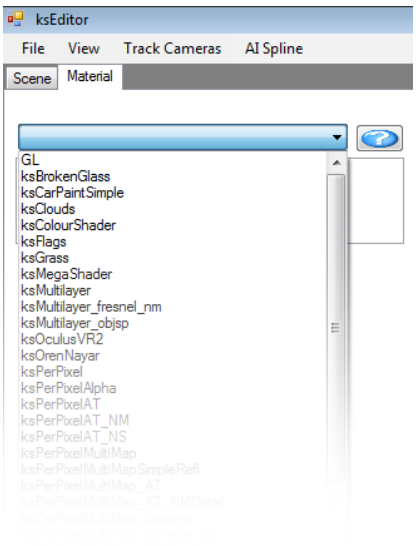
It is important to follow certain rules during car exporting, to avoid errors and extra work, especially regarding material setup.

1) Meshes **must** have only one UV coordinates set.
2) Do not change the material names when you export the car again. The editor always checks for the material name assigned to a shader. In case a car is imported with a name different than before, the shader will not find the reference material and assign wrongly, an empty material.

Once an FBX file is loaded in the editor the materials can be configured.



In the tab **MATERIAL** you can assign a materials to any mesh. Specific material exist, for specific tasks. To assign a material, first select a mesh by right clicking on it The mesh will be highlighted pink. Change the material on the proper slot. By default any mesh is assigned a PerPixel simple shader with default parameters. When a texture is not found or not assigned to a material, a text in the bottom pink tab will show a NULL texture.



Once the car materials are configured and edited, it is time to save a **PERSISTENCE** file. This file contains all the shader modifications made for the FBX file.

The **PERSISTENCE** file is named like the FBX file reference. It is important to not rename the “car_name”.fbx file after the **PERSISTENCE** is saved.
Example: The FBX file **FerrariF40.fbx** , will create a persistence file named, **FerrariF40.fbx.ini**

This workflow allows successive modifications of the car materials even if the FBX meshes and animations have been modified and re-exported many times. Editing will remain possible for every single aspect, as long as the same file names and materials names are kept identical.

The following subjects are under construction.

SHADERS

PRIORITY OF MESHES FOR TRANSPARENT MATERIALS

TEXTURES AND MATERIALS

CAR DAMAGE MESH AND SCRIPT

MATERIALS and DRAW CALLS (wip)

The enhance optimization and achieve better framerate performance,the number of **draw calls** to the graphics engine must be very low. Keeping the number of materials as low as possible is a great optimization technique. A car exterior usually has no more than 10 / 15 materials. The interior is also limited to around 15/20 materials.

For every **material** you can have different **shaders**. The shaders are explained in detailed mode in the SHADER SECTION.

The following **texture feature layers** are supported in game:

DIFFUSE LAYER
NORMAL MAPS (Tangent space and Object space)
SPECULAR MAP
REFLECTION MAP
Exponent Multiplier MAP
MASK MAP (rgb map to define masks for multilayer shader)
ALPHA MAP
DETAIL MAP (a secondary texture layered in various way with a multiplicator of tile)