Battlefield 2 Modding Tutorial 6 - Building the Car - Maya version

by [EA]Lawrence Brown

Download the maya files here

In this lesson we will create a basic car. No guns or anything yet, just a basic vehicle to zip around in.

Note: I've provided example scene files that you should use to follow along. I will be writing this tutorial assuming that you a using these examples. Feel free to try creating your own files if you like, but pay very close attention to the instructions, especially the naming conventions. If you are having problems, you may wish to open one of the existing examples and compare it to the file you are working on.

There is a file called "My_Car.mb" that is the finished version of this tutorial. If you wish to save your work as you proceed, don't overwrite this file because you may want it later. Save your work with a different name, such as My_Car_Test.mb" or something similar.

Set-up:

Before we begin, open the folder "Tutorial Files" that is included with this tutorial. Inside you will find a folder labeled "Vehicles" Place this folder so that the path is as follows:

C:\Program Files\EA G AMES\Battlefield 2\bf2editor\RawData\Objects\Vehicles.

Create any additional folders as necessary to create that path. If you have completed the previous tutorial "Battlefield 2 Modding Tutorial 4 - Building The Bunker", then you should already have the "RawData\Objects" path. If not, then you may h to create the "Objects" folder first.

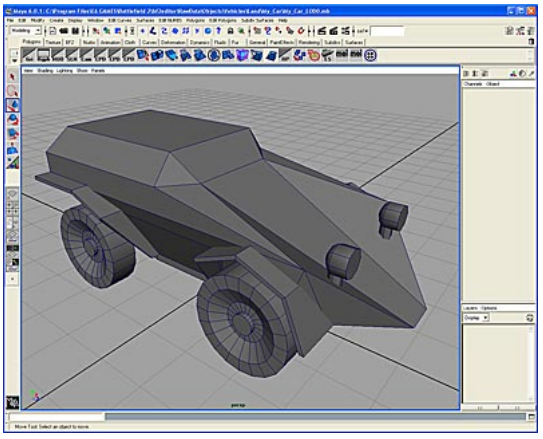
Modeling:

Before we can do anything else, we of course have to model the car. There are many optional pieces you can add to a vehicl such as a cockpit and steering wheel, but for this lesson we're just going to deal with the basics.

The minimum pieces that you need to make a functional vehicle are the following:

Main LOD :

This is the main visible mesh of your vehicle. It should contain the most detail, including such things as headlights, runningboards, exhaust pipes, and so on. This is the model that will be seen when a player gets close to it.



Notice that I've added a few small details, such as headlights and fenders. I've also spent a fair amount of polygons on the wheels. In this case they are quite large (almost 2 meters tall), so more polygons were necessary to keep the edges round. may be able to get away with less polygons on smaller wheels, but keep in mind that nothing makes a model look worse tha wheels that aren't "round". Spend a few extra polygons here and save somewhere else where detail isn't as critical if you ha to.

I've also added some detail to the center of the wheels where the axle is. You can put some detail here, but a lot can be don

with the texture normal map later to add things like bolts. You may also be able to get away with using the transparency channel for things like cutouts in Mag wheels and so on. Like all game models, it's always a tradeoff between detail and engi rendering load, so you'll have to experiment, but keep in mind that this is the highest LOD that will only be seen when a play is close to it. You will make lower detailed versions later on.

Notice that I am modeling the wheels at the same time as the rest of the vehicle. In previous versions of Battlefield, you wo model each piece separately and then assemble them in the con files. In BF2, the entire vehicle is brought into the editor as whole, so if there were things like turrets, guns, hatches, and so on, you would model them all in the same scene and place them in their correct positions. There will be more on this later.

1. Open the scene "My_Car_ LOD 0.mb". The full path to this file will be "C:\Program Files\EA G AMES\Battlefield 2\bf2editor\RawData\Objects\Vehicles\Land\My_Car\ My_Car_ LOD 0.mb" by default.
2. Combine all pieces that are part of the main body (everything except the wheels) into one and give it the name of the vehicle. In this case, name it "My_Car__PlayerControlObject".

The first part of the name is "My_Car" and should be whatever you are going to call the vehicle. The name may contain singl underscores, but no spaces. The second part of the name is "__PlayerControlObject". Notice that there are two underscores a the beginning. This is very important because when the editor reads this file, it will see that double underscore and know tha the next part of the name defines what a particular element is.

The wheels follow a similar naming convention.

3. For the front left wheel, change the name to "My_Car_nav_FL__Spring".

The first part, "My_Car_nav_FL" can once again be anything you like. In this case "Nav" stands for navigational wheel, mean you steer with it. "FL" stands for "Front Left". You can label things any way you like, but it's probably best to stick with these standards wherever possible just to avoid confusion, especially if someone else has to look at your files and so on.

The second part again starts with a double underscore, and "Spring" simply tells the importer that it's a wheel.

4. Name the rest of the wheels "My_Car_nav_FR__Spring", "My_Car_BL__Spring", and "My_Car_BR__Spring".

Notice that the last two refer to back left and back right. The "nav" designation has been removed because these wheels don steer. Remember that the more precise you are with your naming now, the less problems you and others will have later.

Collision Meshes:

Now that we've created our main LOD , it's time to create the collision meshes. You may wish to start by hiding the wheels. won't be using them for now.

Col0:

5. Copy "My_Car__PlayerControlObject" and rename it "col0".

Col0 is the projectile mesh, meaning that all calculations pertaining to projectiles will be done using this mesh.

6. Simplify this mesh by removing all polygons that are not important to projectile collisions. Also remove extra edges in areas that can be simplified.

When simplifying this mesh, think about a bullet flying at your vehicle. Most of the calculations in the game are done determ what strikes what. This is done polygon by polygon, so when you simplify, you have to balance "accuracy" with game performance.

For instance: is it important to know when a bullet hits a headlight? Probably not. Yes, in a perfect world where you had an infinitely fast computer it would be nice to see the effect when a bullet bounces off this small detail, but because the headligl is so small, it isn't really necessary. There's no way a player can hide "behind" the light, and at most angles the bullet will st strike the main vehicle body anyway.

Sure, you might notice this if everything is very still and you are looking at the right place at the right time, but this is a fast moving game. No one will know or care if there is no collision on the headlight, so it should be removed from this mesh. You would do the same for any other small details that stick out.

7. Simplify the mesh further by removing the thin polygons on the edges of the fenders.

We will leave the main top and bottom pieces because they may be big enough to provide cover, but the thin ones are just wasted calculations.

Deciding whether or not to remove fenders can be a bit trickier. Remember that we are dealing strictly with bullet collisions not vehicle collisions. If the fenders really stick out or are very large, you'll have to decide if it is possible for a player to us them for cover. Keep in mind that the game is always in motion, but sometimes static vehicles are used to hide behind, so if the "missing" fenders are obvious enough that the player is going to say "Hey! I was hiding behind this fender and I still got shot!", then you may want to leave them in.

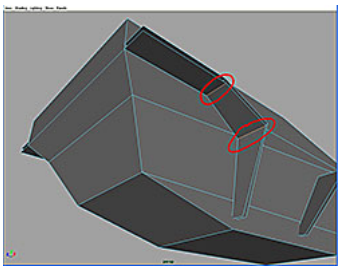
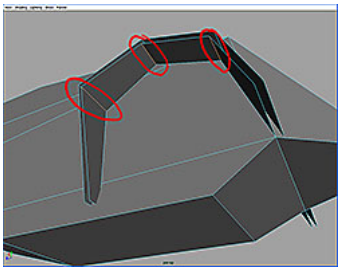
8. There are several optimizations to be made to the main body, so rather than trying to explain every polygon that need be changed, just open up "My_Car_Col0.mb" and have a look for yourself. No need to save the scene you've been work on.
9. Hide everything but "My_Car__PlayerControlObject" and "col0", then move one of them so they are side by side and compare the differences between the two.

Notice that basically what I did was to remove the wheelwells because they were unnecessary details that just slow down th server. I then removed all the extra edges that were left over and removed edges between adjacent polygons that only had small angle of difference between them. I then added back necessary edges to keep the mesh from having any convex polygons. If you study the difference for a while, you'll get the idea.

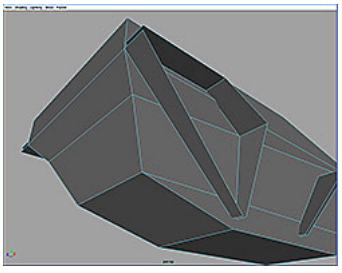
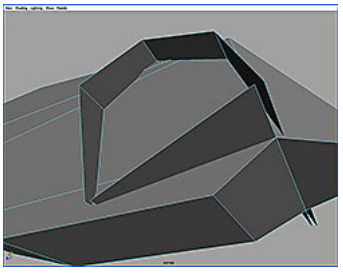
Col1:

This is your soldier collision mesh and is used only for the game to figure out when a player runs into a vehicle, and of cours when a vehicle runs into (or over!) a player. In this case the mesh is so simple already that there will be very little optimizir necessary.

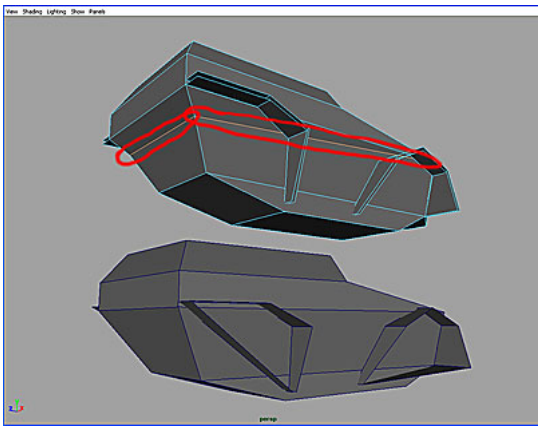
10. Move the pieces back to the origin and then hide everything but col0. (or just reopen the scene)
11. Copy col0 and name it "col1".
12. Remove the following edges and delete any excess vertices if there are any:

The result should be this:

13. Additionally, remove these edges:



Note the before and after in this shot.

Make sure you do the same to both sides of the vehicle so that it remains symmetrical.

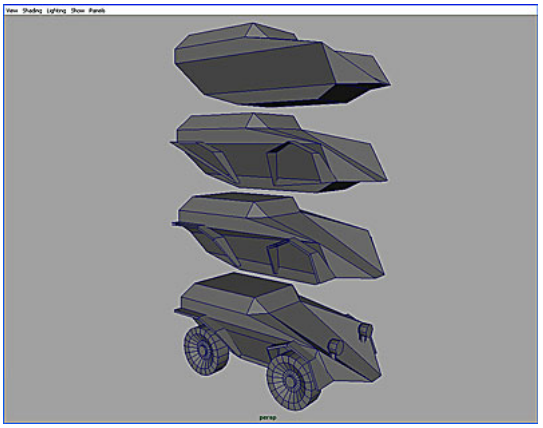
Col2:

You have now finished the soldier collision mesh. The last one to create is col2, the vehicle collision mesh.

The vehicle collision mesh is used only for calculating impact between vehicles and should normally be the simplest mesh of Anything sticking out like fenders or antennae should be removed and the body simplified as much as possible.

14. Copy col1 and rename it "col2".
15. Remove the fenders.

You should now have 3 collision meshes, plus the main LOD and Wheels (I've spread them apart here just so you can see all the pieces):



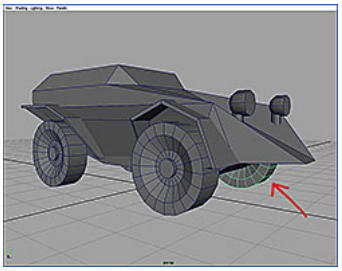
Wheel Collision Meshes:

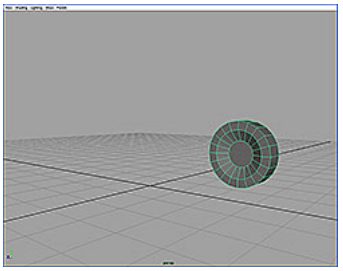
We've still got one more set of collision meshes to make. These are for the wheels. Unlike the body, there are only 2 collisic meshes for the wheels and they work slightly different.

Wheel Col0:

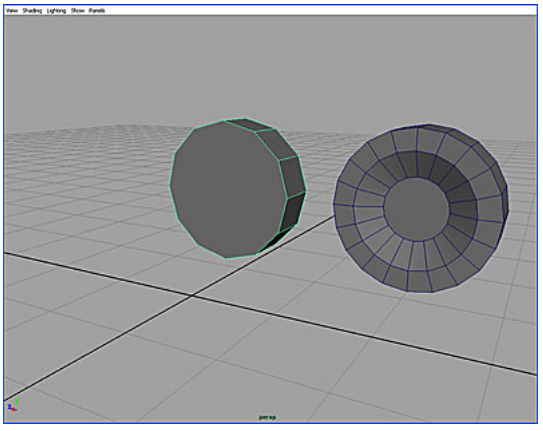
This is the bullet collision mesh.

16. If you've moved any of the pieces around, make sure they are all back at their correct positions.
17. Select the front left tire, then hide unselected.





18. Create a simplified version of the wheel by either copying it and removing the excess edges or by creating a new cylinder and shaping it accordingly. The object is to have a cylinder that is the same dimensions as the wheel, but with few polygons as possible.



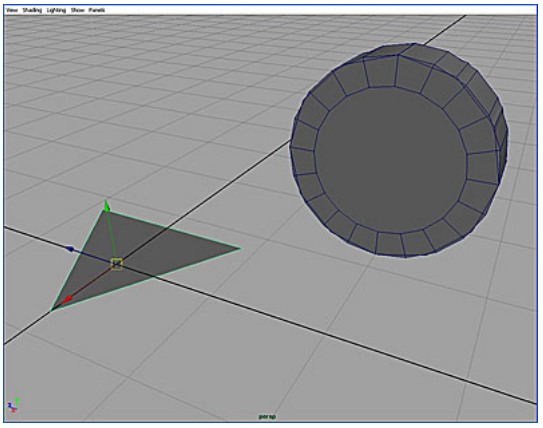
Here I've moved the new cylinder away from the wheel so you can see them both. You will want to line up the new cylinder the wheel so they both match. I find the easiest way is to display the rotate pivot of the wheel (Display>Component Display>Rotate Pivots), then snap the cylinder to it. (Selecting the rotate pivot option from the menu turns it off again.

Don't try to name the new piece yet. It won't work right now because of Maya naming conventions.

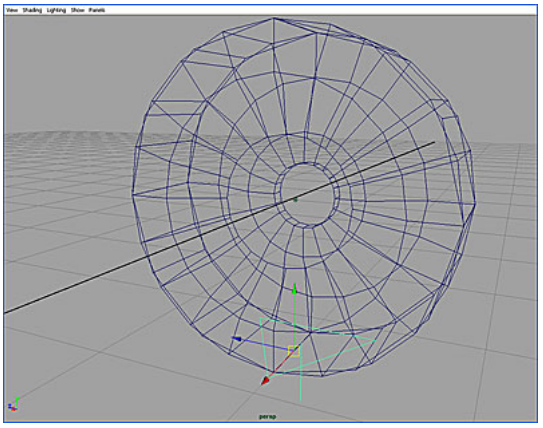
Wheel col1:

The last collision mesh we have to make is col1 for the wheel. This mesh acts a little differently in that it is only used by the vehicle "engine".

19. Create a single, three-sided polygon. You can do this any way you like. One way is to create a cylinder, change the number of sides to three and the SubdivisionCaps to 0, then delete all but the bottom polygon and center the pivot poir In any case, what you want to end up with is this:



20. Move this polygon to the bottom center of the tire, the place where the tire normally touches the terrain. The size of t polygon isn't important, so you may wish to scale it down to the size of the tire.



In this screenshot, I've also displayed the normal of the polygon. Notice that it is pointing down! All calculations are basicall done from this normal, so if it is facing up, your vehicle won't work correctly.

Again, don't try to name the new piece yet. It won't work right now because of Maya naming conventions.

21. Do this for the remaining 3 tires.

Texturing:

We've modeled all the pieces now. Before we assemble them together we are going to want to do some basic texture mappi

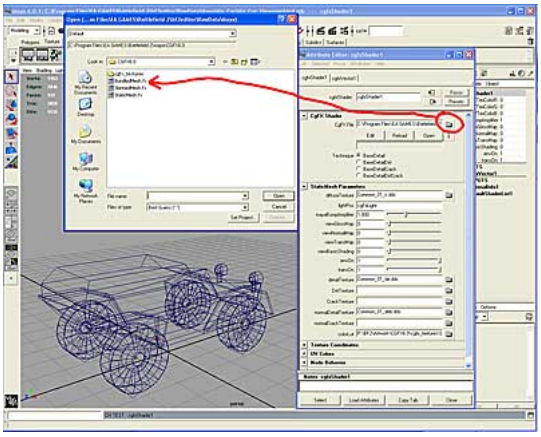
You may continue with the file you've been working on or optionally open "My_Car_Untextured.mb" to follow along.

The reason we have to apply textures is because the importer uses the names of the shaders to determine that this is a vehi and also uses the names to determine which materials to apply to what surfaces.

The reason we have to do this before we assemble everything together is because it is difficult in Maya to select objects in a hierarchy and work with them. If we do the work now while we are still dealing with individual pieces, it's a lot simpler.

There is so much information already in this tutorial that I'm not going to add to that with a full lesson on texturing. Instead, I'm just going to show you the bare minimum that is required to properly prepare the vehicle for import into the editor.

22. In Maya, select just the pieces that will be visible. This will be the piece labeled "My_Car__PlayerControlObject" and tl 4 visible meshes of the wheels, which are labeled "My_Car_FL__Spring" and so on. We don't want to select any of the collision meshes, just the ones that will be visible while playing the game.
23. With these pieces selected, type "dnyUVPanel" (or click on the shelf button if you made one) to open up the custom U\ editor window.
24. In the "DiceNY UV Editor" window, press the "Assign" button. This will create a new cgfx Shader and assign it to all th meshes that are selected.
25. Open the hypershade and double-click on the shader that was created to open up the attribute editor. It will probably l labeled "cgfxShader1".
26. In the "CgFX Shader" section of the attribute editor, click on the folder icon to the right of the field labeled "CgFX File'
27. In the window that opens, select "BundledMesh.fx" and click "open":

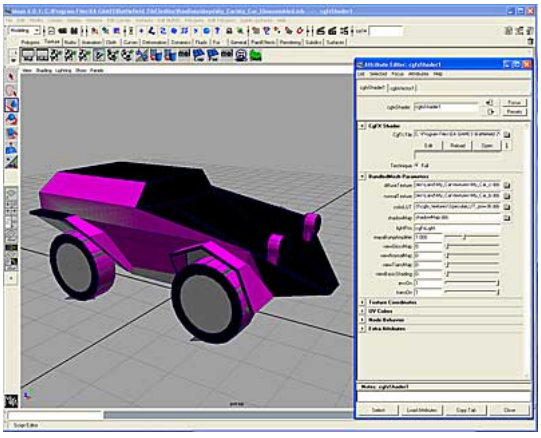


This is necessary because the importer reads the name of the shader to determine what type of object it is importing. "BundledMesh.fx" get's used on anything that the player can interact with, such as vehicles and static weapons.

Once this fx shader is selected, the options in the attribute editor will update.

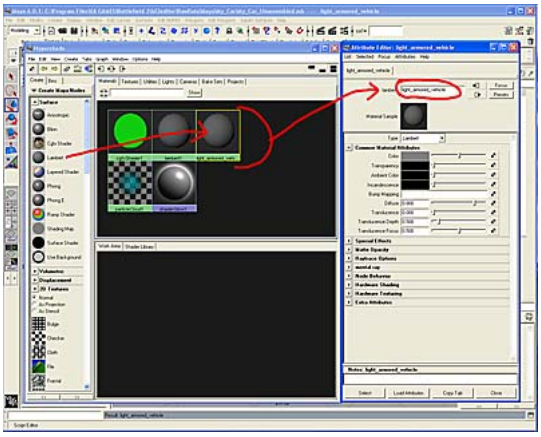
28. In the "BundledMesh Parameters" section, click on the folder icon labeled "diffuseTexture". In the window that opens, select "C:\Program Files\EA G AMES\Battlefield 2\bf2editor\RawData\Objects\Vehicles\Land\My_Car\textures\My_Car_c.dds". This is the diffuse color texture for the ca
29. Now click on the folder icon next to "normalTexture" and select "My_Car_b.dds" from the same folder. This is the norr Texture for the car. (the "b" stands for "Bump")
30. Finally, click on the folder icon next to "colorLUT" and load the texture "C:\Program Files\EA G ames\Battlefield 2\maya\C G FX6.0\cgfx_textures\SpecularLUT_pow36.dds".

You don't have to worry about what's in the "shadowMap" field. You should now see the hot-pink car of death:



We've set up a basic texture for the visible mesh, but we also need to apply textures to the collision meshes.

31. Hide the visible LOD meshes that we just textured so that we can work with the collision meshes.
32. In the Hypershade, drag a new Lambert shader to the Materials window, then double-click on it to open up the attribut editor and change the name to "light_armored_vehicle":

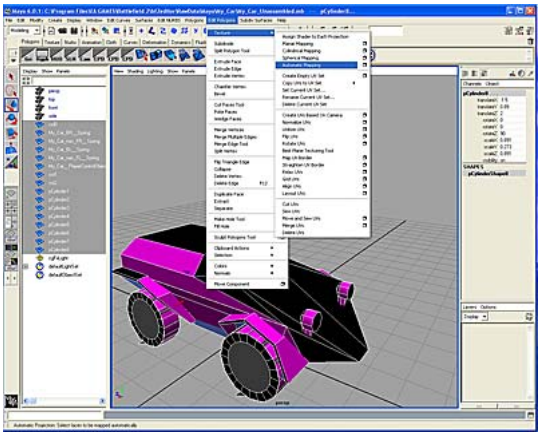


The color of the shader doesn't matter, so you can change it to something more identifiable if you like. What's important is h it's labeled. "light_armored_vehicle" must be spelled correctly because the importer will look at the name and assign that material to all polygons it's applied to.

33. Select the three collision meshes of the main body. These would be labeled "col0", "col1", and "col2". (Don't select an wheel parts yet.)
34. Apply this shader to those three collision meshes. You may now wish to hide them because we are going to do the wheels next.
35. Create another lambert shader and label it "Rubber".
36. Select all the wheel collision parts and assign this shader to them. (In case you didn't hide them earlier, make sure yo are not selecting the visible meshes of the wheels that we textured earlier, only the collision meshes.)

You have now created and applied all the shaders. We still have a little bit of work to do before we can start assembly.

37. Unhide all pieces of the car and select them all. This means both the visible and collision meshes of all parts.
38. With all the pieces selected, select "Edit Poygons>Texture>Automatic Mapping" from the main menu. We don't care w the options are:



The reason we did this is because the importer (and the game) don't like polygons that have zero surface area. This is the quickest, easiest way of making sure that all the polygons of the model have their UV's laid out correctly. You won't see muc change because we are using solid colors, but if you open the DiceNY UV Editor again, you can see how all the polygons have been laid out.

Triangulating the meshes:

There is one more thing we have to do before we can begin final assembly, and that is to triangulate the meshes. The import and game don't like polygons with more than 3 sides, so it's easiest to fix this now.

Note: Before triangulating, it's always a good idea to save a copy of your scene file. This way if you have to come back and rework something, you don't have to deal with all the extra edges that can make things difficult. If you want to come back to this file, save it now. Use any name you like, just remember what it is.

39. Select all meshes in your scene.
40. Select "Polygons>Triangulate" from the main menu.

Your meshes should now all be triangulated.

Note: the screenshots that follow will not look exactly the same as your model. The following screenshots do not have the shaders applied and the meshes are not triangulated. Don't let this confuse you. You should be working with the triangulated

textured version of the car from this point on.

Final Assembly:

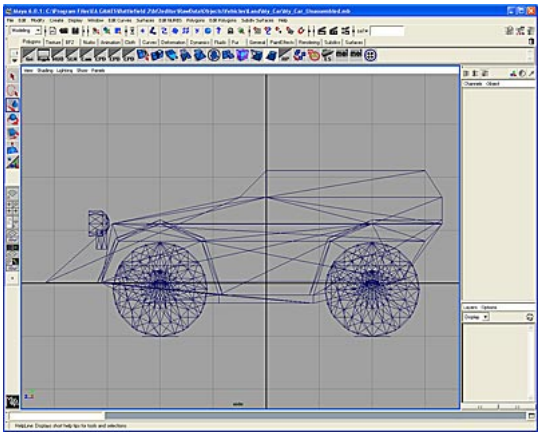
You've now created all the major pieces necessary to make a basic vehicle. The last steps are to assemble everything in a hierarchy, create a few bits necessary for the exporter, and do final prep work on everything.

Setting the Center of Gravity:

Most physics calculations are done from the pivot point of the main LOD , so we have to adjust this now.

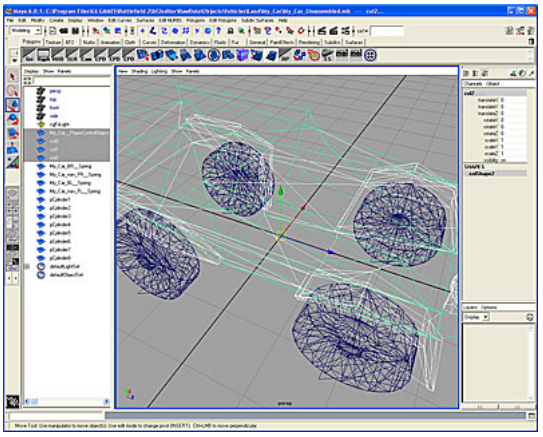
For these steps, you may continue with the scene you've been working on. You may also open "My_Car_Unassembled.mb" if you prefer.

41. Select all objects and move them down so they line up as shown here. Notice how the grid origin isn't at the bottom o the wheels anymore. The vehicle is a bit more centered:



Finding the right center of gravity for each vehicle will take some practice and a lot of experimenting, but a good place to st is normally somewhere between the bottom and center of the vehicle.

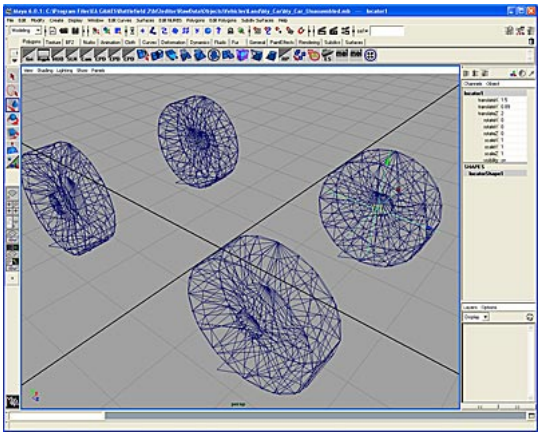
42. Select the LOD and col0, col1, and col2. (Not the wheel collision meshes)
43. Move their pivot points to the origin, freeze transforms, and delete all history.



Setting Wheel Positions:

The positions of all pieces of an object, such as wheels, rotational bundles, turrets, and so on, are dependant on their offset from the origin, so we now need to set the positions of the wheels.

44. Hide the LOD and collision meshes for the main body to make working with the wheels easier.
45. Select the visible LOD for the front left wheel and make it's rotate pivot visible if it isn't already. (Display>Component Display>Rotate Pivots)
46. Create a locator and snap it to the rotate pivot. This is just a temporary locator that will be used in a moment.



47. Select the wheel and its two collision meshes and group them. (Do not include the locator!)
48. With this group still selected, move it's pivot point to the locator.
49. Move the group to the origin and freeze the transforms. (Make sure each piece has its transformations frozen, not just the group node.)
50. Move the group back to the locator we created earlier. Do not freeze transforms on these pieces again! The importer needs to read the offset to know where to put the pieces.
51. Ungroup the pieces of the wheel.
52. Create a cube at the origin and scale it down so it's very small but still pickable.
53. Freeze the transforms on this new cube.
54. Rename the cube "My_Car_nav_FL__RotationalBundle".

Note: You may have to do the renaming in the outliner because the channel bar doesn't like long names, or just come up wit shorter name. The important part is the "_RotationalBundle" part at the end.

55. Move the cube to the locator for the wheel.

This cube is just a placeholder. When the car is imported into the editor, the name of the cube tells the editor "create a rotationalBundle here."

Note: At this time nothing should be grouped. You may want to open your outliner and check to be sure there are no groupin and that there are no extra "unused" nodes.

Setting up the Wheel Hierarchy:

56. Select the pieces we are using as collision mesh 0 and 1 of the wheel we are working on and group. (Just the collision meshes, not the other pieces.)
57. Move the pivot point of this group to the locator.
58. Rename the group "nonvis_". (The single underscore at the end is necessary.)

Note: any time you are working with collision meshes, they must be grouped this way with this name.

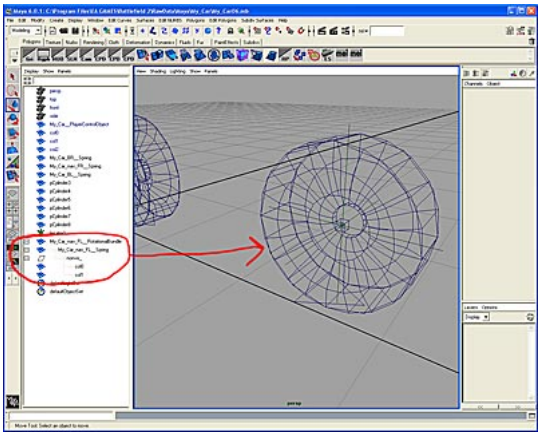
59. Parent the "nonvis_" node to "My_Car_ FL__Spring".
60. Parent "My_Car_ FL__Spring" to "My_Car_nav_FL__RotationalBundle".

Renaming the Wheel Collision Meshes:

Now that the pieces are in a hierarchy, Maya will let you rename pieces with duplicate names. (as long as they aren't in the same branch)

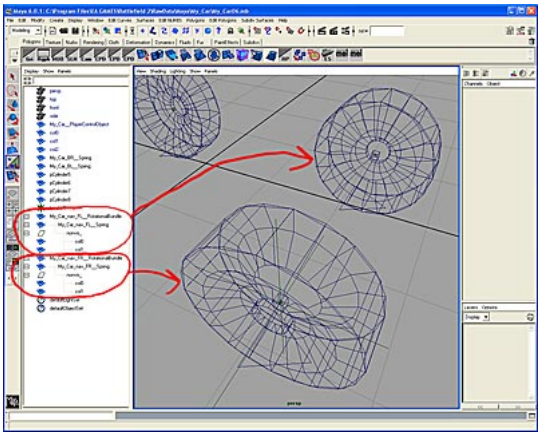
61. Select the low-poly cylinder we are using as the wheel collision mesh 0 and rename it to "col0".
62. Select the triangle polygon we are using as the wheel collision mesh 1 and rename "col1".

I know this seems complicated at first, but after you look at it a few times, it will make sense. If you followed all the instructions, your front left wheel should have a hierarchy that looks like this:



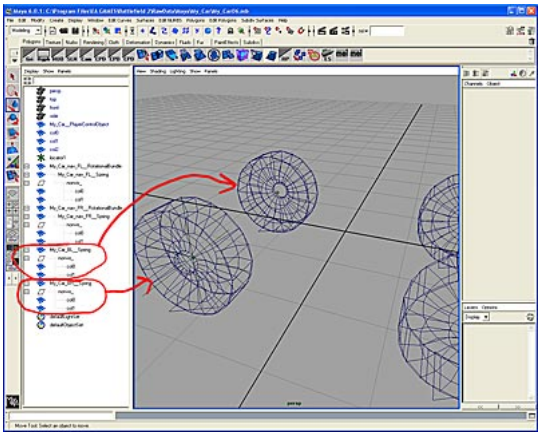
63. Repeat steps 46-63 for the right front wheel. Instead of creating another locator, just move the one from the left whee to the right wheel. Also, anywhere you used "FL" before, make sure you change it to "FR" for "front right".

Once you've finished, your hierarchy should now look like this:



Note: It may be hard to see in the screenshot, but the front wheels are not connected to each other. They are two separate, identical hierarchies.

64. Now do the same thing with the back wheels. The only difference is that you do not make a cube with the name endin "__RotationalBundle". The back wheels don't steer, so they don't need the rotational bundle. The hierarchies for the ba wheels should look like this:



Creating the Engine:

The wheels are set up, but now we have to connect them to an engine.

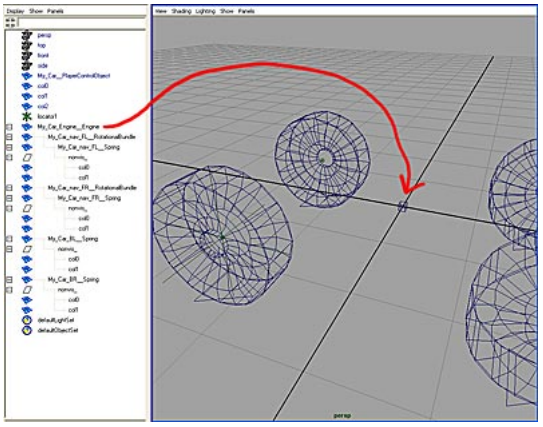
65. Create another cube, scale it down, and freeze its transformations. (I generally scale these cubes to .1)
66. Rename the cube "My_Car_Engine__Engine"

The first part of the name, "My_Car_Engine", can be anything you want. Following our naming convention, it starts with the name of the vehicle and adds "Engine" so that we can tell what it is. All underscores in this part of the name must be single.

The second part of the name is "__Engine". Since it starts out with a double-underscore, the importer will know what this is a create the correct part in the .con file. In this case of course, it's an engine.

Connecting the Wheels to the Engine:

67. Parent the top node of each wheel hierarchy we previously made to the "engine". Your hierarchy should now look like this:



Note: the order of the items in the hierarchy isn't important as long as the connections are correct. I've moved the various pieces around so that they are listed in order. What is important is what is connected to what.

The Final Connections:

We're almost done. We just have the main hierarchy left to create.

68. Unhide all the pieces of the vehicle body.
69. Select col0, col1, and col2 of the main body and group them.
70. Again, name this new node "nonvis_"
71. Parent this "nonvis_" node to "My_Car___PlayerControlObject".
72. Also parent "My_Car_Engine__Engine" to "My_Car___PlayerControlObject".
73. Select "My_Car___PlayerControlObject", which should be your top node right now, and group it so that an empty node created.
74. Name this new group node "lod0".
75. G roup "lod0" to itself and name the new node "geom1".
76. Move the locator we created earlier to the origin. (Or create a new one if you deleted the other one.)
77. Rename the locator "root_bundledMesh".
78. Parent "geom1" to "root_bundledMesh".

Creating the First-Person View:

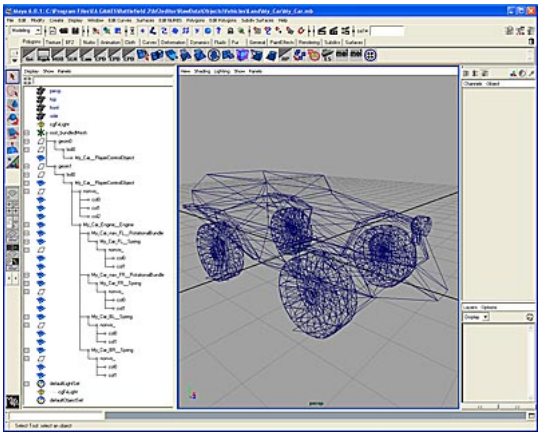
" G eom1" is the set of meshes that are used for the third-person view, which is what you see when you are not in the vehicl or when you are in the vehicle and have switched to one of the chase cameras.

We still need to create the first-person view, which is " G eom0". This is also known as the Cockpit. We're not going to do m with it, but if we don't create it now, the importer will give us an error because it's expecting it.

79. In the hierarchy you just created, select " G eom1" and duplicate it. This will create a new node labeled " G eom2" in y hierarchy along with all the pieces that are under it.
80. Rename this new node to " G eom0".
81. expand this node and the pieces under it Until you can see "My_Car__PlayerControlObject.
82. Delete everything in the " G eom0" branch until all you are left with is " G eom0>lod0> My_Car__PlayerControlObject"

This is a quick way of creating the first-person geometry. We don't need any of the collision meshes, the engine, or anything else, just the mesh that you will see when inside the vehicle. When you get to creating a more finished vehicle, you would actually model things like the dashboard, windshield, hood, and so on, and delete anything on the "outside" that you couldn't see from the driver's point of view.

Your hierarchy should now look like this. Notice that I've moved the " G eom0" branch up to the top of the hierarchy for the sake of organization:



You've now completed all steps necessary to import a basic vehicle into the editor.

Note: the reason there are nodes like "lod0" and "geom1" is because there are additional components that will be added late but you don't have to worry about this for now.

The file "My_car.mb" is this same completed file, ready for import into the editor. Don't overwrite this file. If some of these steps were confusing, you may want to open the finished file to compare it to yours. This file may also be used as reference making other vehicles.

You may now continue with "Battlefield 2 Modding Tutorial 7 - Importing and Editing the Basic Car".