Preface

I will go over how models are loaded in sm64 from start to finish. This includes how data is loaded, how it's stored and how the game identifies what data belongs to what model. At the end I will wrap it all up by going through an example on how to load your own model. If you just want practical examples and dont care to understand what's going on then skip to the section called Quick Breakdown.

Raw data

The raw data for a model has two parts, vertices and textures. Every texture format is stored in a different way. What matters is the bit size and format of the texture. Generally you will see texture formats displayed as the color type, followed by a bit size. So RGBA16 means RedBlueGreenAlpha color space, with 16 bits per texel. CI4 means ColorIndexed 4, which refers to 4 bits worth of colors, or 16 indexed colors total. The other formats are Intensity, IntensityAlpha, and YUV which is never used (The only one that isn't an acronym). If you change texture types you have to change everything else in your display list to match. So know which texture type you're using.
Vertices are more simple. They're 16 bytes each, and store the X, Y, Z position. The S and T texture coordinates(vert and horiz), and the X,Y and Z normals, or RGBA vertex color values. Each value has its own way of being stored which if you're interested you can read about in the programming manual.

Display Lists (F3D)

A display list (sometimes referred to as Fast 3D), is the code that takes the raw data, and connects it all together. From the DL it is processed then sent to the RCP (a chip in the N64 dedicated to drawing triangles) to be rasterized.
Display lists define explicitly what will be displayed on the screen. This is the most direct control you have over the image. I will not go over all of f3d as it is quite complicated, but I will give a brief overview of just how a triangle is drawn.

First, the DL tells the hardware to sync. This is because there are two different chips working together, so they have to sync or the data won't be drawn correctly. Next, various effects are added or set up. There are a lot of extra graphic effects you can do that I wont go over, just know they are initialized before triangles are drawn.
After that, there are 4 important DL (Fast 3D) cmds you should know that go before every textured triangle.

- 0xFD - load texture. This loads the texture at a defined address with a specified format. If you ever change this, make sure the format matches your texture type.
- 0xF2 - set texture size. This sets the dimensions of your texture (e.g. 32x32 or 32x64 etc.) If you make it smaller, then your texture will be cut off part way. Making it larger than your texture does nothing.
- 0xF5 - set tile properties. This cmd basically controls how the texture will appear compared to its actual coordinates. You can mirror, clamp, or shift the texture in either direction with this cmd.
- 0xFC - set color combiner. This cmd has the most control over what colors go into the triangle. There are many graphics effects, and this controls how they combine to form the final pixel color on the triangle.

If you ever change a part of a model and it doesn't look right (the texture is all black, too dark, too light, is cut off halfway) then these are the most likely culprits.
After all the setup of graphical effects, the display list gets to drawing the actual triangles. This is done by loading vertices with an 0x04 cmd (up to 15 at a time) then drawing a triangle with 3 verts using an 0xBF cmd. These cmds are the meat of most F3D.

Geo Layouts

Almost every DL in game is loaded from a geo layout. A geolayout basically sets up how the DL should be loaded. This can be from scaling it, translating/rotating, adding a shadow or setting up render distance.

We are only interested in loading the display list right now. This can be done many ways but right now we will focus on the two most common.

- 0x15 - load model
- 0x13 - translate and load model

Both load the DL from a defined address, and then set the layer the DL goes into. The second byte is the layer number, the last 4 bytes are the address. Each layer has a specific purpose, here is a brief overview.

- 0 = unused, free for custom use
- 1 = solid, will be most of your tris
- 2 = solid details drawn on top of layer 1
- 3 = unused, can put custom things here (originally for intersecting tris)
- 4 = alpha textures (completely transparent or opaque)
- 5 = transparency (entire texture is partly see through, NO Z BUFFER)
- 6 = alpha decals (layer 2 but for alpha, NO Z BUFFER)

When you change a DL, change the layer to match. Most commonly you will use layers 1, 4 and 5.

Model loading

Models are finally loaded in the level script. The difference between a model, and a display list is that a display list is just data that is drawn. A model is a defined structure of data loaded to be displayed in a certain way. While most of the time a model is just a loaded display list, it can also be multiple display lists strung together using a geolayout, or even no data (an invisible model).

A level script is what defines how the level is set up, and each level script loads every single model used in the level. They are loaded using an 0x22 or 0x21 cmd. The cmd sets the model ID, and has a defined address of where to load the data from. The difference between the two is just what kind of data it loads. 0x22 loads a geo layout, 0x21 loads a display list directly. There's not much to these, they just define what data to load.

Quick Breakdown

Now that each piece of the process was explained, I'll give an overview of the process from start to finish.

- A model is loaded in the level script. It's given a model ID, and a place to load the data from. It can load a geo layout or a display list.
- In a geo layout, display list(s) are defined and given a structure on how they are transformed. The actual display lists are loaded with either a 0x15 or 0x13 cmd.
- In a display list, raw model data is processed and set up to be drawn on screen. The color in each triangle is configured, then the triangles are drawn.
- The raw data is loaded from a display list. It holds textures and vertices.

The basic idea of loading a model is to follow this structure. Now depending on what you want to edit in a model, you know where to look. I will go over some examples now.

Editing models

Ex. 1 Changing a model ID

To change a model ID is quite simple. Model IDs are defined in the level script, in a 0x22 or 0x21 cmd. To find the level script, you should probably use Quad64 (beta build 5 is the latest version). This is the only tool I know that has a script viewer that formats it in an easy to read way.

To use it, load the rom, then go to the level you want to change the model ID of something in. Open script dumps (Misc->script dumps) then find the model load cmd. The model ID will be in hexadecimal. Once you find it, just open a hex editor, go to the rom address of that cmd, then change the 4th byte, which is the model ID.

Ex. 2 Changing a texture.

To change a texture, you have to edit the DL. Specifically the F3D texture loading cmd. As I mentioned earlier, there are 4 commands you should keep in mind. To edit a texture you will usually have to change at least 2 of them.

Keep in mind that while you can change the defined texture size, the texture coordinates will not change. So if a triangle only shows 32 texels of a texture. It will show that no matter the size of the texture.

First, we have to find the texture we will replace. I will be replacing the red ! box side texture from a 32x64 RGBA16 texture to a 16x8 IA4 texture. The easiest way to find a texture is to use Quad64 script dumps. Place the model inside your level, and then open the textures tab. If you click the texture you want to replace, it will have a segmented address next to it. Write down the address of texture, then do the same for your replacement texture. Next go to script dumps, and navigate to the objects tab. You will then find the object you placed in the level. The name should match whatever name is in the object list.

Once you find the object, click the fast 3d tab and search for the 0xFD texture load with the address you wrote down earlier. In a hex editor, replace it with the new texture. Change the format to match as well. You can find formats here:

F3D reference

Next we will have to change the 0xF2 and the 0xF5 cmds to match. These should be directly under the 0xFD cmd in our quad64 script dump. Following the reference we have, change the parameters to match our new texture's format.

Finally we should get a good result.

What you may notice is that the ! box had a jump in its f3d. This is because the boxes reuse data to save space. If you look at another color box you will notice that box is now messed up. This is something you have to be careful of when making these types of changes.

Ex. 3 Changing a single display list

Here we want to change the display list loaded in a geo layout. I will change mario's shoe, to be his hand making a peace sign.

You can find mario's geo layout by giving an object model ID 1 (mario's model ID) then navigating to script dumps and clicking the geo layout tab on your object with ID 1.

As you can see, there is a lot of stuff in the geo layout. Luckily, mario64 has been decompiled and we can view the source of mario's geo layout:
Mario's geo layout
Looking at decomp is a great resource for more complex actors when you don't know what display list corresponds to what data. Most animated actors in sm64 have complicated geo layouts so I recommend using decomp to help find the correct part. Then once you find the right loading cmd in the geo layout, you can find it in quad, then the rom.
If you search, you will see that line 85 has the peace sign hand. Lines 139 and 154 are mario's shoes. To find the equivalent lines you could count the lines in the quad64 script dump or copy paste the entire dump and then open it in an editor that counts lines. Once you have the quad script dump lines laid out, you have the display list addresses of his shoes and his peace sign hand. Now all you have to do is put the peace sign DL address in his shoe geo layout.

And finally the result.

If you inspect mario's geo layout more you'll see there are more definitions for his shoes. Those are for different cap effects and low poly mario, and the different combos of them. I won't replace the DLs there but if you want to actually replace everything be sure to replace the models there as well.

RAM Banks

So far, I have only explained that there are addresses for the data we are loading, but not what those addresses actually are. To understand that, you need to understand how RAM banks work and how data is loaded. I have written a tutorial on this topic you should read it before continuing with this tutorial here:
Hacking Tutorial Preface

After you read that, you should understand what a RAM bank is, and how to find the ROM location from it.

RAM banks are defined in level scripts. There is a special sequence in the level script that loads all the banks used in that level, which is a string of 0x17 cmds. The 0x17 cmd defines the bank number, ROM start and ROM end locations. This means if you want to load more data, you can change these cmds to make space for it in RAM. You can only load so much extra data though, so be careful (you'll know when you load too much because the game will crash).

So in order to define a custom model, you normally have to have space in a RAM bank. You should know how to find ROM addresses from bank addresses so now all that's left to do is import the model.

Ex. 4 Loading a custom model

When I load a custom model, I extend two banks. One bank to hold the fast 3D DL data, another to hold the geo layout data. You shouldn't put them in the same bank because as mentioned in the preface tutorial, DL data can only go from bank 0x1 to 0xF, so space is limited.

****Note- backup your ROM before doing any level script change****

Bank 0x4 and 0x17 are the easiest to extend. They are used for mario, so they're in every level (you can use bank 0x7 for level specific models it works the same way). Great for loading objects in every level. Using script dumps, find the 0x17 cmds for bank 0x17 and bank 0x4. The last 4 bytes are the end ROM location. Next you should find the actual space you want to extend the banks to. Simply goto their current end location, and scroll down until you either have enough space or you run into other data. Typically bank 0x17 will never be relocated, but bank 0x4 will. This is because geo layouts are much smaller than display lists.

If you don't have enough space in bank 0x4 (a common issue) then you have to relocate it. To relocate a bank, first find a spot in the ROM where you do have enough space. In a 64mb rom this is usually at the end of the rom (I like using 0x3000000+). Then copy

from the start to the end address using the locations listed in the 0x17 cmd for bank 0x4. Finally paste that to the location you chose.
Next you will have to change the addresses in your 0x17 cmd. Change the start and end cmd to match the location you moved it to, or if you didnt move it just change the end location to the extended position. Now what you should do, is write down the ROM and bank positions of the start, original end, and new end of your bank. You will have to reference this often for calculating addresses for the various places your import to.
Now, you will have to prepare a model to import, and the model importer. The best custom importer is the model importer module that bundles with rom manager. It should also be a stand alone program you can launch in the same folder.
There are 4 fields you have to fill to import a model. ROM start, RAM start, geo pointer, and col pointer. Max Length just exists to prevent overwriting other data, which we solved earlier by moving the bank.
Now from the data you wrote down earlier you should have two of these already. ROM start is the end of the original data, or the first spot with no data there in your bank. This should be shown by having all 01s repeating or 00s repeating. From here you should be able to calculate the bank offset easily. Those go in ROM offset and RAM offset inside the model importer respectively. Next is geo pointer. This is now when you have to set up the geo layout. If you don't know anything about it, I will provide a template one for you here.

- 20 00 xx xx -replace xx with render distance
- 04 00 00 00
- 15 xx 00 00 -replace xx with your layer.
- 00 00 00 00 -this will be your geo pointer. Put the start location of this as your geo pointer in the model importer.
- 05 00 00 00
- 01 00 00 00 -end geo layout

This loads a model inside a layer, and sets up how far away it will be rendered from. Put this in bank 0x17, and make sure you extend it so that it is all inside the bank.

Collision pointer is the last option. This is for solid objects. If you're making a solid object, then you should have a behavior script prepared already (I won't go over these). Put the collision pointer as the last 4 bytes of the 0x2A behavior cmd.
If you don't have collision, on the model importer, make sure you uncheck collision. You should be good to import now.
Finally you need to load your model. You need to get the bank address of the start of your geo layout, and then go to your level script. Using quad64 you can locate the 0x22/0x21 cmds for your level. Now you can either define a new model, or replace an existing one.

Replace existing model

Simply find the model load like I explained in example 1, and then replace the last 4 bytes with the start address of your geo layout. To test if it worked, save the rom in your hex editor, reload the rom and level in quad64 then create an object with that model ID. It should display properly if everything went correctly. If an error shows up you either wrote down the wrong pointers, or did not extend the banks properly.

Add a new model

You can only do this in a custom level. Scroll down to the model loads in bank 0x19. Go to the last one, and you should see some other cmds after it. Copy everything after it until padding (copy at least an extra line of repeating 01s) and move it forward 8 bytes. Then in the new space, write your model load cmd.

- 0x22 08 00 xx aa aa aa aa

xx is the model ID you want, and aa is the bank address of your geo layout. Again, save the rom in your hex editor, reload in quad and add an object with that model ID to check if it was added properly.

Conclusion

You should now have the knowledge needed to replace almost any model, or display list. If you are new to this, it's very likely you will mess up a lot, so be careful, write down all your numbers and make many backups.