Technical Reference Manual

Star Trek The Next Generation Starfleet Command III

Modeling, Modifying, And Adding Ships

taldren

The Art & Science of Gaming

Introduction

This manual is intended to give you a solid background in what's needed to create the best possible ship models for Starfleet Command III. The Taldren art team uses 3D Studio Max for ship modeling, and Adobe Photoshop for painting ship textures, and this manual will concentrate on our use of those tools. You may use other applications to accomplish many of the same tasks, but we can only document what we do ourselves. We'll assume that you have a basic familiarity with those two applications as we go on here; if you're new to either program you may find that you have to look up some operations in their own manuals.

The Starfleet Command series of games uses a 3D file format called MOD. That format has been used since Starfleet Command I, and the same geometry of 3D models can be used in any version of the game – but because we have added new effects in successive versions, there can be problems when using an SFC II generation model in SFC I, and so forth (though SFC I models can "go forward" they require additional work to take advantage of the newer game features like illumination maps). The best practice is to modify your models for use in a particular generation of the game. Section 1.04 describes the steps to follow if you want to convert an earlier model for use in SFC III.

Because of significant engine changes in SFC III, it is more important than ever that you customize existing models for this new generation of the game.

We'll start with an overview of 3D models for Starfleet Command III, explaining what's changed, and then expand on the entire process of building, texturing, and optimizing your models for their best performance.

After that we'll tell you how to integrate your new ship model into the game.

PART ONE

CREATING THE ASSETS FOR YOUR SHIP

1.01 Overview: Realtime 3D Models and SFC III

1.01.01 Background

3D modeling and animation have been with us for about two decades now. In the last ten years more and more games have used "realtime" models and engines rather than "pre-rendered" graphics for most parts of gameplay.

"Pre-rendered" imagery relies completely on the output of a professional rendering system that is optimized to give the best possible results. A program like Discreet's 3DS Max allows artists to create many types of object geometry, texture them with image maps and procedural textures, add effects, and animate the results. These renderers provide great image quality at the expense of speed. Artists can choose to do things that take quite awhile to render, because ultimately no one will know how long it took. Just how well it turned out.

Realtime 3D engines do the same basic tasks, but are optimized for speed. Many features like procedural textures, motion blur, antialiasing or special effects are either unavailable, or are available only in a much more limited form. The bottom line in a realtime game engine is that it has to pump out frames at a fast enough rate that its animation is as smooth as, we hope, what you'd see on television or at the movies.

Because of these differences, good modeling practices are far more important in realtime than in pre-rendered graphics. The polygon counts of models are very important; the size and number of their image maps is important; the correct structure of the models is important. What all these things mean is that discipline is important. While many game players begin 3D modeling because of games, they're starting in a branch of computer graphics that's much less forgiving than the work we also do in pre-rendered animation.

Starfleet Command III models can take advantage of these types of texturing and features:

Geometry – compatible with earlier SFC's
Color Maps
Illumination Maps
Progressive Damage Maps (with their own Illumination Maps) - NEW
Specular Intensity - NEW
"Break" Models – NEW explosion ordering
Dynamic Level of Detail (LOD) - NEW
Schematic Images in the game user interface – NEW
Ship/Hull Name in the game user interface – NEW
Ship/Hull Description in the game user interface – NEW
Ship Hard Point diagrams in the game user interface – NEW

1.01.02 Geometry (and file naming restrictions)

The 3D object geometry of the .mod format is compatible with all versions of Starfleet Command; however, some texturing features will break models when they move to earlier versions. See sections 1.02.01 – 1.02.07 for more on model geometry.

In addition, the game now counts on certain names being the same (other than their file extensions). The model's directory, model files, text files, and other images must use the same "root" name. This is detailed later, but an overview of the file naming conventions is this:

Folder name: Dispeptic_Gorilla
Model name: Dispeptic_Gorilla.mod
Break model name: Dispeptic_Gorilla_brk.mod
GF file name: Dispeptic_Gorilla.gf
Text file name: Dispeptic_Gorilla.txt
Tactical Schematic image: Dispeptic_GorillaTI.bmp
Beauty Shot image: Dispeptic_GorillaBS.bmp
Vessel Library schematic image: Dispeptic_GorillaVL.bmp

Image maps and illumination maps (see below) are included in the model's materials, in Max. No file naming convention is necessary, but as a rule we have named illumination maps <texture name>_i.bmp. Damage and Damage Illumination maps are also named to match their parent textures as described below, in sections 1.02.08 – 1.02.13.

1.01.03 Color Maps

These, which Max calls "Diffusion Maps", are the most basic type of image maps. They just "paint" color on the textured surface.
See sections 1.02.08 – 1.02.13 for more on textures and materials.

Limitations:
Does not need to be square, but must be power of two in each dimension (usually 128 x 128, 256 x 256, 512 x 512, or 1024 x 1024; but also 256 x 128, 512 x 256, etc.).
Can be either 8 bit or true color.

1.01.04 Illumination Maps

These correspond to the Color Maps, making certain areas of the ship look bright (completely lit) regardless of the direction of the light source. The effect is that those areas of the ship's surface are actually luminous. Note that there is no "glow" that extends beyond the surface of the ship, but an artist can paint a glow in the textures that seems to light up the surrounding details of the model.
See sections 1.02.08 – 1.02.13 for more on textures and materials.

Limitations:
Does not need to be square, but must be power of two in each dimension (usually 128 x 128, 256 x 256, 512 x 512, or 1024 x 1024; but also 256 x 128, 512 x 256, etc.). Usually corresponds to the size of the matching color map.
Can be either 8 bit or true color – but there's really no point in these greyscale images for using more than 8 bit color.

1.01.05 Progressive Damage Textures- NEW

The new progressive damage system applies a "damaged" color map and illumination map to the site of a weapon's impact in the game for greater realism.
See section 1.02.12.

Limitations:
Does not need to be square, but must be power of two in each dimension (usually 128 x 128, 256 x 256, 512 x 512, or 1024 x 1024; but also 256 x 128, 512 x 256, etc.). Usually corresponds to the size of the matching color map.
Can be either 8 bit or true color – illumination maps should really always be reduced to 8 bit.

1.01.06 Specular Intensity - NEW

This unmapped effect (meaning that it is not controlled by an image) gives an adjustable level of "shininess" to the highlights of your model. Each model can have a unique amount of Specular Intensity.
See section 1.02.13.

1.01.07 Break Models

These are alternate versions of the ship model that have been split into pieces. The pieces explode away from each other when a ship is destroyed.

NEW: parts of ships blow off at the beginning of an explosion now. The break model chunks numbered 1 through 4 are selected randomly as the first piece to go, at the start of an explosion.
For more on break models, see section 1.02.07.

Limitations:
Needs to be in the same model directory as the ship model with the same name plus "_brk", like "RagingPenguin.mod" and "RagingPenguin_brk.mod".

1.01.08 Dynamic LOD- NEW

In earlier versions of Starfleet Command, a .mod file contained three different versions of the ship. These used a naming convention to distinguish three levels of detail. Typically, two or three sizes of textures were also used.

In SFC III, the game reduces detail on models dynamically as they recede from the camera. Only one version of the model is needed (not counting the _brk model).

To see how object geometry affects Dynamic LOD, see sections 1.02.01 – 1.02.04. There are also ways to optimize your object's materials for Dynamic LOD. This is described in section 1.02.09.

1.01.09 Schematic Images in the Game User Interface- NEW

It is now possible to have your user-created ship model show up in your user interface panels. This is described in sections 1.03.01 – 1.03.03.

1.01.10 Ship/Hull Name and Description in the Game User Interface- NEW

Text files in the model's directory can contain the name and a description for your new ship. These appear in the Vessel Library and Ship Config screens when your ship is viewed. While this is meant for an "in character" description of the starship, there is no reason why you could not add a credit to yourself (the modeler) or a history of changes to the model, if you've kitbashed someone else's work.

Polite modelers give credit where it's due. Impolite modelers should remember that polite modelers could be armed.

For more about adding text to the game's user interface, see section 1.03.04.

1.01.11 Ship Hard Point diagrams in the game user interface – NEW

Hard Points, in the SFC series, determine the location of weapons and other systems on a ship. In SFC III these represent empty mounts that may or may not be filled. Hard points are indicated in a model with the use of dummy objects (not new) but now, since a player-created ship can appear in the user interface, there's also a way to place the hard point icons that appear in the interface. This is detailed in section 1.03.03.

1.02 Creating and Exporting SFC III Ship Models Using 3D Studio Max and Photoshop

1.02.01 Object Geometry

Because you convert a polygonal "mesh" to .mod format at the end of this process there aren't any limits on what modeling tools and modifiers you use up to that point. The SFC exporter will convert your object and its material to .mod format.

The SFC III art team has a responsibility to create models that work well on a variety of computers – many of them older and not what you'd call state-of-the-art. This means that we have to be very careful about the amount of memory and processing time our models consume when they appear in the game. We have upped the ante a bit in SFC III, but we are always faced with these constraints – constraints that you don't necessarily have to share, unless you also want every user to be able to see your work the way you intended for it to be seen.

In SFC III we have tried to avoid using more than 1500 polygons in any non-"break" version model. Very many of our models use under 1000 polygons. The break version of a model always has a somewhat higher poly count than the unbroken version, because of its internal breaks and two sets of caps for them (the segment on each side of the break has a cap).

As I said, you don't have to share these constraints, but by exceeding them you do exclude some people from using your work. And remember that not everyone who downloads your models will understand the inner workings of the game as well as you do, and they may wonder why their game no longer works the way it did.

In SFC I and SFC II the game engine was quite robust when it came to less-than-ideal geometry, like duplicate vertices, doubled polygons, and intersecting polygons. SFC III's dynamic LOD system is much less forgiving, though. While models that exhibit these problems may still work pretty well in-game, there are likely to be some visual glitches like flashing polygons, and dynamic LOD will not work well with these models. It may not work at all.

Why do you care whether dynamic LOD works? Well, maybe you don't. But the players who download and install your models will, even if they don't know what the problem is.
Any Level of Detail system exists in order to improve performance. As an object gets farther from the game camera, you can drop unnecessary detail because it just isn't visible any more. If you don't bother to support dynamic LOD your models will not perform well when they're used. And there's this: wouldn't you rather spend a performance hit on additional detail in the model, so that it looks even better all the time?

So let's look at some examples of what works well, and what doesn't.

For best results:

1.02.02 Seamlessness

A model should be one seamless object – meaning that there are no gaps in it anywhere, so that it encloses one contiguous empty space on the inside. There should be no intersecting polygons, either.

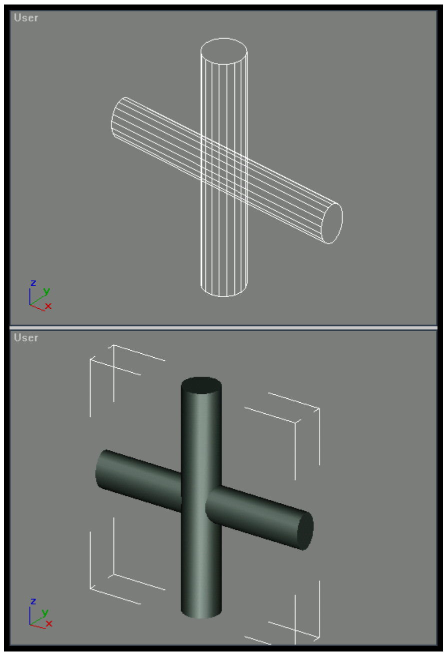


Figure 1.02.02a shows an object made of two intersecting cylinders. Although they are joined together as a single object, they are not "seamless": their polygons intersect in the middle, and they do not enclose a single contiguous space.

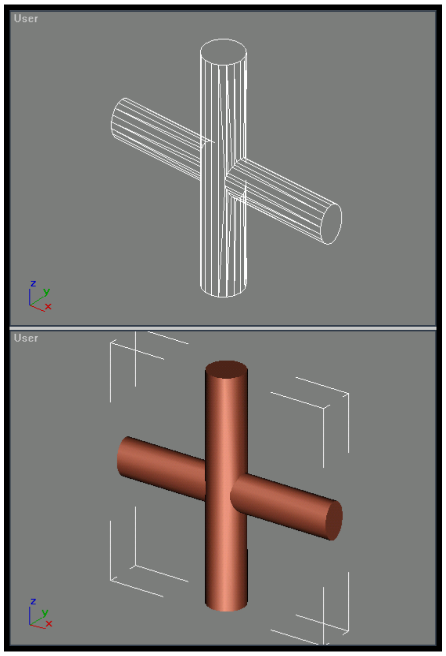
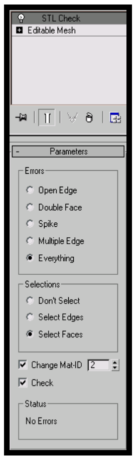


Figure 1.02.02b shows an object that looks just the same, but does not have intersecting polygons, does enclose a single contiguous space, and therefore is "seamless". It was produced by performing a Boolean operation (Compound Objects/Boolean) on the two original cylinders.

One way to achieve this is to run Max's Boolean operator (Compound Objects/Boolean) on two intersecting parts to make a seamless whole out of them. This will increase your polygon count, since new polys will be created at the seams. After performing a Boolean operation it is often necessary to tidy up the joint between the sections by welding vertices, and sometimes by reassigning the smoothing groups of the model.

The easiest way to check for these conditions is to run Max's STL-Check modifier. This modifier was intended to check objects for seamlessness and other errors, so it's pretty much exactly what we're looking for. You can have the modifier highlight or change the Material ID of problem areas.



STL Check
Editable Mesh

Parameters

Errors: ( ) Open Edge, ( ) Double Face, ( ) Spike, ( ) Multiple Edge, (●) Everything

Selections: ( ) Don't Select, ( ) Select Edges, (●) Select Faces

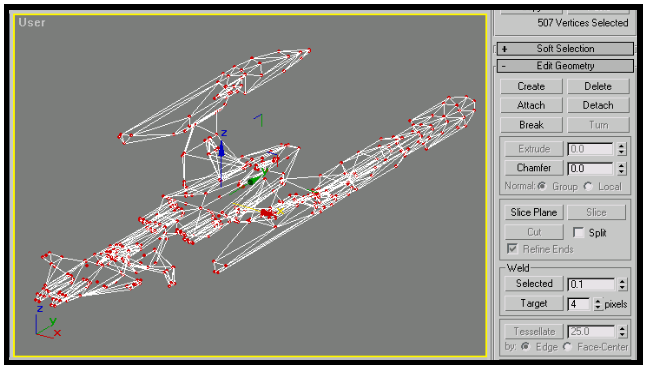
☑ Change Mat-ID 2

☑ Check

Status: No Errors

Figure 1.02.02c: The STL-Check Modifier in 3D Studio Max

Note that this modifier has some trouble adapting to changes you make after it's run, so to be safe, make your corrections, delete the modifier from the stack, then run the modifier again.



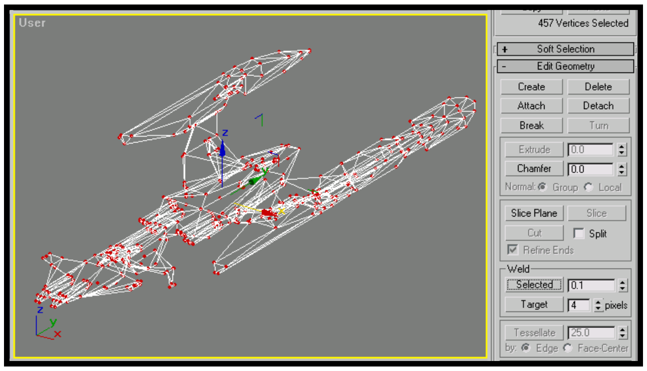
507 Vertices Selected

Weld: Selected 0.1; Target 4 pixels

Figure 1.02.03a shows a ship model with all its vertices selected: note, in the upper right, that there are 507 selected.

1.02.03 Eliminate Duplicate Vertices

A model should contain no duplicate vertices (that is, no two points should occupy the exact same location in the model).



457 Vertices Selected

Weld: Selected 0.1; Target 4 pixels

Figure 1.02.03b shows the same ship model after "Weld Selected" is performed, using a Threshold of .1 units. Note that now, at the upper right, we have only 457 selected. 50 duplicate vertices have been eliminated.

This is an error that commonly occurs when two parts are joined together without welding their duplicate vertices.

The easiest way to eliminate duplicate vertices is to select all the vertices in the model and then Weld them with a threshold of .1 – this should eliminate all vertices that share the exact same location in 3D space. You might need to redo your smoothing groups after you perform this step.

In other cases, where vertices are very close but not in the exact same spot, you may need to select and weld them a pair at a time. To do this, select only those vertices you want to weld, and set the Threshold to a large number like 1000 to ensure that they do. Max turns them into a single vertex midway between the two, which is not always what you'd like; so you might need to move the welded vertex afterwards.

1.02.04 Eliminate Duplicate Polygons

A model should not have any duplicate polygons (polygons that share the exact same location). This is almost impossible to spot without running STL-Check, but can sometimes be detected because it creates smoothing problems on a surface.

To fix duplicate polygons you really need to select and delete one of them, often flip the other, and finally, redo your smoothing groups.

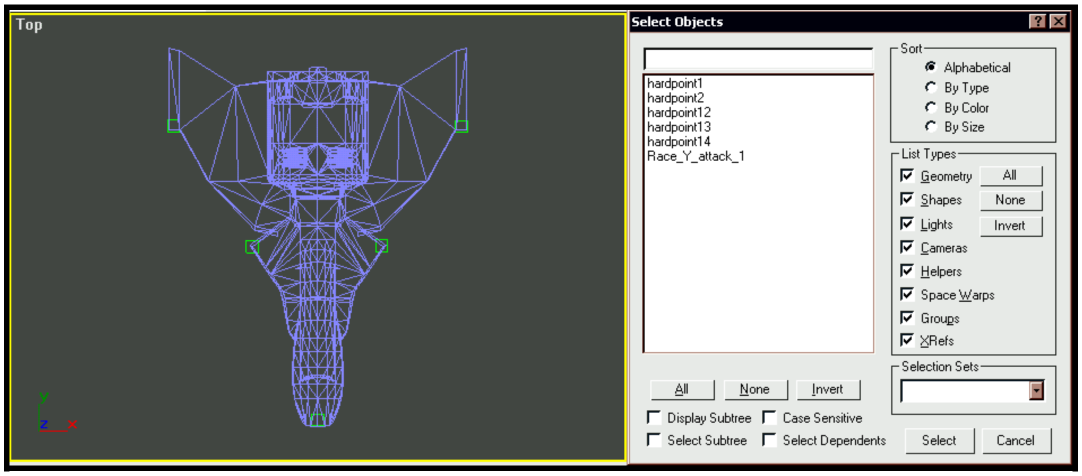
These steps are most complicated when you're adapting existing models – when you're building a ship from scratch you can keep the basic principles in mind as you work, and minimize the number of corrections you need to make to your object geometry.
The benefits of creating "error free" geometry are that your model will perform well in the game, look good in the game, and not do any harm to the game's performance – especially on older machines. Also you get this warm, fuzzy, karmic glow. Usually around 4 AM, when you're finally done.

Even in pre-rendered work these are good modeling practices. They speed up rendering and prevent some really odd problems in Max's viewports.

1.02.05 Hard Points

We use dummy objects in Max to represent the hard points of a ship (locations of weapons and systems mounts). The name of the dummy object tells the game what hard point it represents.

Hard point names correspond to information in the specfile for the game, where tactical ship data is kept.



hardpoint1
hardpoint2
hardpoint12
hardpoint13
hardpoint14
Race_Y_attack_1

Figure 1.02.05a shows a ship model (blue), its dummy Hard Point objects (green), and the Max Select List displaying the names of the Hard Points on this ship.

In SFC I and II, weapons hard points were associated with numbers. There wasn't a distinction between mounts for primary weapons and heavy weapons. That was all determined in the specfile.

In SFC III, the game needs to understand whether a particular hard point is a mount for a primary weapon or a heavy weapon. We manage this by reserving some numbers for heavy weapons, and others for primary weapons.

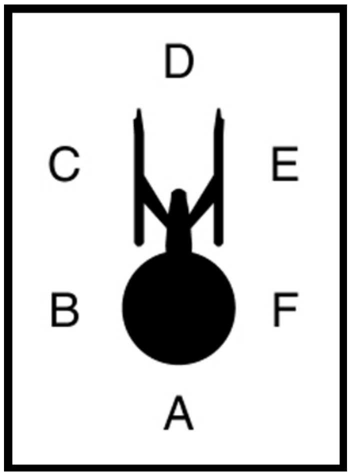
In Max, the dummy objects are named:

"Hardpoint1" – "Hardpoint 11"	:	Primary Weapons
"Hardpoint12" – "Hardpoint25"	:	Heavy Weapons

Systems hard points are important to the hard point layouts that appear in the game's user interface (discussed in section 1.03.03), but they don't appear as dummy objects in the model itself.

1.02.06 Damage Points

Damage points, like hard points, are represented in the model by dummy objects.



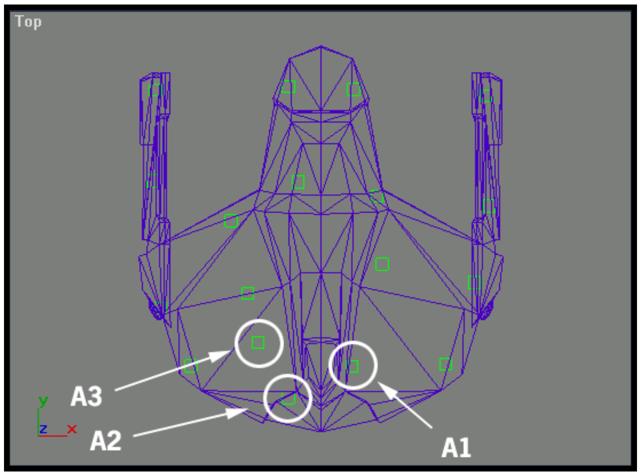


Figure 1.02.06a shows a simplified diagram for damage point placement. The three "A" points go at the front of the ship, and so on, clockwise around. Each letter gets three points (a1, a2, a3, etc.). Figure 1.02.06b shows a top-down view of an actual model with its damage point objects (in green). The three A points are indicated.

These aren't locations of "real" things on board. They just serve to give hints to the game about where to damage a ship. Don't get excited; this is just about how the damage is shown visually in the game – the actual ship damage doesn't depend on these points.

The game decides what part of the ship is to be damaged, then looks for the damage points in that area. There are six sets of three damage points each, and the game will choose one of them when the weapons effect strikes the model.

So you get to place your damage points in the area that is indicated by the diagram – the game may do some very odd things if you place these on the wrong side of the ship! – and select what parts of your ship you'd like to get blasted.

1.02.07 Break Models

A break model starts its life as a textured ship model. It's quite difficult to retexture a break model to match its unbroken version, so you should always wait till you're sure your model's done before you break it.

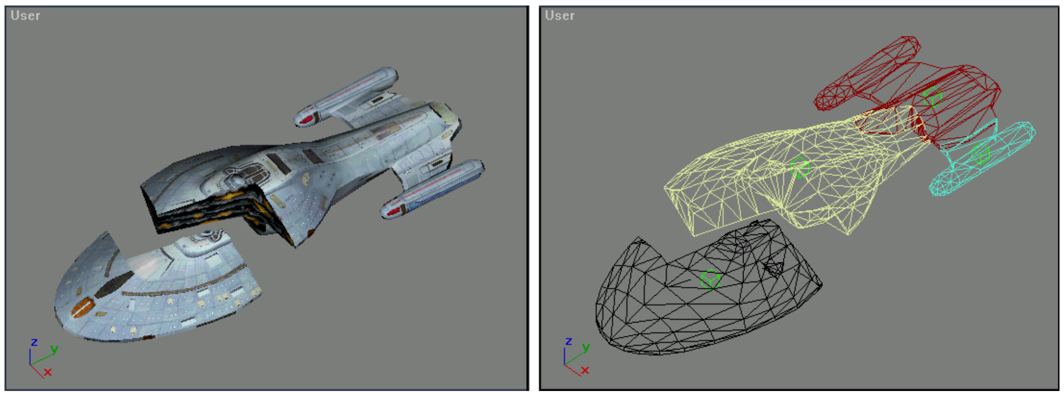


Figure 1.02.07a shows two views of a break model; we've moved one segment out of place to expose the damage cap on one segment. In the wireframe view, you can see the individual parts in different colors. The green boxes are the pivot point dummy objects.

We make break models by loading the ship, positioning planes or other shapes to bisect it, and then create a Boolean division using those parts. Max's Booleans are easily confused, so it can be safest to do this one break at a time.

Because they're not perfect, it's usually necessary to seal some gaps after breaking. The "Cap Holes" modifier can be useful for this, but in the end there's no substitute for getting your hands dirty and stitching in a poly at a time. You also add a lot of polygons at intersections when doing Booleans, so it's worth your while to take a good look at the results and weld some vertices where needed. When that's all done, you may need to adjust some smoothing groups on the model.

If you can't stand Booleans, you can also select sections of the ship by polygon and Detach them. The advantage is that it's simpler and results in a lower poly count; the disadvantage is that it's often pretty obvious that you've just separated along polygon edges, rather than creating a more natural looking break. You seal the caps in just the same way, whichever method you choose.

Add new UVW coordinates to the "caps" at the breaks. Our textures usually include some fiery exposed decks or other structural details for the damage caps, though they fly by fast enough that we never know if anyone notices them.

Each chunk of the break model is numbered. Chunks 1-4 may blow off first (the game randomly picks one of those four at the start of an explosion) so make sure that those particular chunks make visual sense when they blow. For example, if a chunk in the center of the Enterprise's saucer blew off first, that might look odd; a warp nacelle, or a chunk on the edge of the saucer section, would make a lot more sense.

Because the .mod exporter will renumber pieces based not on their name, but in *their order of creation*, there is a separate step you'll need to perform to guarantee that the fragments are ordered the way you want them. We'll discuss that in a moment.

Each chunk has a corresponding pivot point which acts as the center of the chunk's rotation while it spins off into space. The position of this pivot point will have an important affect on how that looks – if the pivot is all the way at the end of a piece, the piece will spin around that end. Sometimes, you might want that to happen.

When they're created in Max, break model chunks are named "<shipname>_brk_1" and so on; pivot points are named "object01pivot" etc., the numbers corresponding to the chunks they're paired with. In fact, it doesn't seem to matter what precedes "01pivot" or "02pivot", etc.

As we mentioned above, the .mod exporter will number these chunks in the order they were created—not necessarily in the same order you name them. There's a workaround we use to ensure that we really get what we want.

1. Go ahead and build the break model and number its chunks in the order you want them to be named—parts 1 through 4 will be the pieces that you'd like to be seen blowing off first, while higher numbered chunks are the big central bits of the ship that you want to restrict to the final explosion.
2. Create a matching set of pivot points for the chunks, and number them to match their corresponding ship parts. That is, pivot object 01 is somewhere in the middle of ship fragment 01.
3. Save this Max file as, for example, "<shipname>_brk.max".
4. Delete the ship parts, leaving only the pivot point objects in the scene. Save this version of the file as "<shipname>_brk2.max".
5. From the File menu, select "Merge", and pick <shipname>_brk.max as the file to merge in.
6. Pick the 01 chunk of the ship model from the list of objects, and merge it into the file.
7. Repeat this for each chunk, until they're all present. Their order in the scene now matches the numbers you assigned to them, and all will be well. Export this version of the break model to .mod format.

The final result is a ship that actually looks unbroken – it's been broken "in place", so all the parts nest right up next to one another. Use the SFC exporter to save it out as "<ship name>_brk.mod", in the same directory as the original model. (See section 1.02.14 for more on the import/export module).

1.02.08 Texturing Your Model

The SFC III art team has a responsibility to create models that work well on a variety of computers – many of them older and not what you'd call state-of-the-art. This means that we have to be very careful about the amount of memory and processing time our models consume when they appear in the game. We have upped the ante a bit in SFC III, but we are always faced with these constraints – constraints that you don't necessarily have to share, unless you also want every user to be able to see your work the way you intended for it to be seen.

Most of our textures are 256 by 256 pixels. There are exceptions (our planets, for example, now use 1024 x 1024 textures, and the Borg Cube uses 512 x 512), but that's generally our rule. We always reduce our textures to 8 bit color.

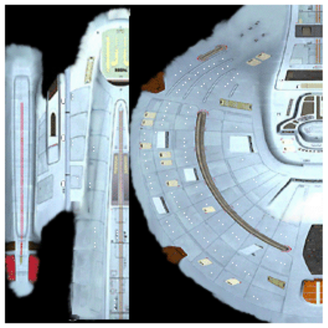
As I said, you don't have to share these constraints, but by exceeding them you do exclude people from using your work. And remember that not everyone who downloads your models will understand the inner workings of the game as well as you do, and they may wonder why their game no longer works the way it did.

For textures specifically, one way to work around this would be to create more than one version of your textures. Once Max's material is set to use an image, changing its size or color depth doesn't undo the assignment – so the same model can use more than one set of textures, provided the texture names are what the material expects. So you could zip up an alternate texture pack for some users. That way some players could use high-resolution, true color textures while others could use a lower resolution 8 bit version. All you'd need to do would be to convert and/or scale the images and make them available in more than one way. The key thing is that the names of the textures in each set should be the same, and of course it's whichever set is in the model directory that will be loaded by the game.

Remember that illumination maps never benefit from more than 256 colors (shades of grey), so it's wasteful to distribute them as true color images.

Our models are all created using Multi/SubObject Materials, so that multiple image maps can be used by the same model. The SubMaterials contain Color (Diffusion) and Illumination (Self-Illumination) mapping channels. Other mapping channel information will probably be converted but may cause problems in the game – for best results, do not assign textures to the other mapping channels (such as bump, specular intensity, etc.).

Finally, the game engine will load the following formats: .bmp, .dds, .dib, .jpg, .png, and .tga. You will most likely have used .bmp files while creating the model in Max, but if you have some reason for wanting to distribute the work in another format, you can convert the texture to one of these. Keep the entire name the same, except of course for the file extension, and the game will load it as though it were the original .bmp file. Keep only one set in the model's directory.



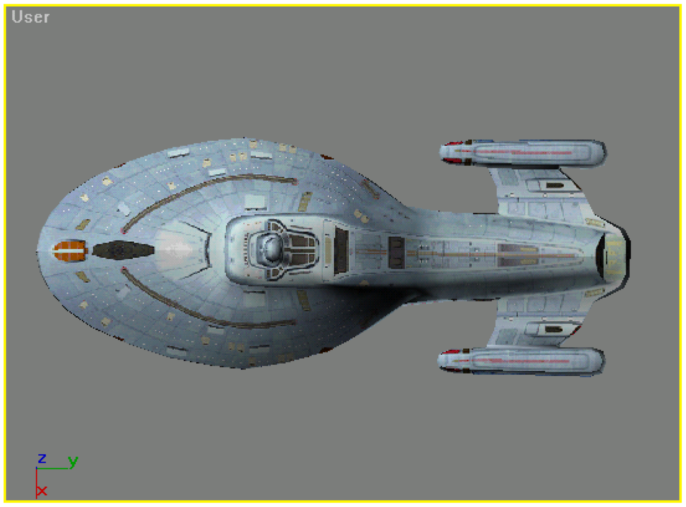


Figure 1.02.09a shows a texture image and the ship it's used on. Note that both parts of the texture are mirrored, so that their reverse image covers the opposite side of the ship. This presents a problem to the Dynamic LOD system only along the line where the texture "flips".

1.02.09 Optimizing Materials for Dynamic LOD

We've already seen how a model's geometry can help – or hinder – the game's dynamic system for levels of detail. Depending on the way you've mapped your model, texturing may also have an effect.

Because we're often trying to save texture memory, when a ship is symmetrical we may "mirror" its textures from one side of the model to the other. That way our image maps use only half the memory as they would, had we textured both sides uniquely.

One characteristic of the dynamic LOD system is that if two adjacent vertices have the same UVW coordinates *and the same material ID*, the system becomes upset and confused about what to do with them. Strange things begin to happen on the surface of the model. The solution is to create a copy of the original sub-material. Assign that copy to all the polygons on one side of the model, leaving the other side's polys assigned to the original material. Repeat for any texture that is mirrored across adjacent polys.

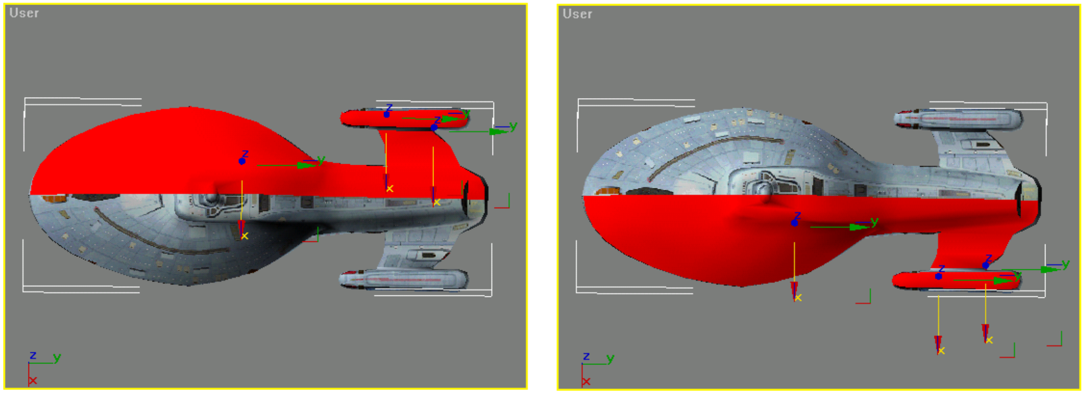


Figure 1.02.09b shows the same model, modified so that Dynamic LOD works properly. On the left, those polygons with Material ID 4 have been selected (red). On the right, those polygons assigned to Material ID 8 have been selected. These two Materials are exact copies of each other, so the ship looks just as it did before. But because the Material IDs have been changed, that mirror point along the centerline no longer confuses the Dynamic LOD system.

The result? The mirrored texture appears where it did before and everything looks the same to the eye; it's just that two materials have been used, not one, to get the same result. Dynamic LOD heaves a sigh of relief and works again.

1.02.10 Color Maps

In Max, set up the Diffusion mapping channel normally. This will be converted and used by the game, and the result will look the same as it did when viewed in Max.

1.02.11 Illumination Maps

In Max, set up the Self-Illumination channel normally. This will be converted by the game, and the result will look the same as it did when viewed in Max.

Remember that only 256 levels of grey will be used by the illumination map, so there's no point in using more than 8 bit color for this channel. It saves memory. It's the right thing to do.

1.02.12 Damage Maps (and Damage Illumination Maps)

Damage maps are not set up as mapping channels in Max. They are not referred to by the material for the object.

They work because their names are identical to their matching texture but with the letter "D" appended to the end – like "LemurFrigate_1.bmp" and "LemurFrigate_1D.bmp". When the Lemurs start taking damage, that "D" texture is automatically mixed in at the point of impact. The illumination maps for those same textures would be something like "LemurFrigate_1_i.bmp" and "LemurFrigate_1_iD.bmp".

As far as their appearance goes, Damage maps are based on the corresponding color maps but show an all-over blast scar, with some glowing bits. The texture tends to look like a glowing charcoal bricquet. Artists need to be careful about painting large, realistic hull ruptures (or any very large features) because you cannot know what parts of the damage texture will be mixed into the surface; the soft edges of the damaged areas mean that any large painted feature - like exposed decks - may end up fading out to the undamaged texture in strange and unplanned ways. We've tried to keep exposed hull structures fairly small, in the hopes that they'll all be blasted into visibility at once.

Here's the process we use for creating a damage texture from a color map. The steps are pretty simple, and the first few stages produce a good background texture for damage without a lot of work.

You may want to create a special version of your model that uses the damage textures as its Diffusion channel, so that you can see how the damage is shaping up. No player will ever see all the damage at once, so it's okay that the ship will begin to look like something from "Night of The Living Dead".

These steps assume that you are using Adobe Photoshop in a 5.0 or later version. With other tools, you'll need to improvise a bit to get similar results.

1. Copy the original texture and promote it to RGB color (Image/Mode/RGB Color).

2. Duplicate the texture into a new layer.

3. Run the "Filter/Texture/Craquelure" filter on this new layer. Adjust it till its scale suggests a pattern you can use as a beginning for your damage.

4. (Optional but recommended) Undo the Filter, then mix part of its effect in using "Filter/fade Craquelure". Most iterations of the filter are stronger than what you want, which is the suggestion of a lot of rippled, crinkly cracks over the surface of the ship. It's just the starting point.

5. Create a new layer on top of that one. Select black and use the paintbrush tool, with the smallest (first) brush, to scribble

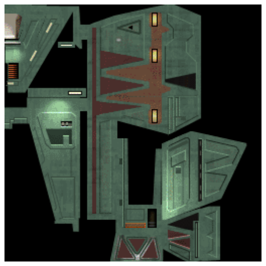


Figure 1.02.12a: The original texture

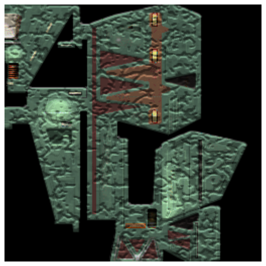


Figure 1.02.12b: After the (faded) Craquelure Texture Filter

over the background, as shown. Try to scribble more in the crevices of the texture, like at points where two plates are joined on the hull.

6. When that layer's been thoroughly scribbled, duplicate it; run one of the blurring filters on it (either "Blur More" or "Gaussian Blur"). This creates a softened version of the scribbled layer that also darkens the overall tone of the texture. Keeping the unblurred version visible, adjust the opacity of this layer until the combination of layers seems right to you.

7. Compare this image to the undamaged version. You want to make sure that the damage texture is much darker than the undamaged version. For lighter surfaces (like Federation ships) you may need to use Image/Adjust/Brightness & Contrast to darken the first layer. When you're satisfied with the overall tone of the image, you're done with the automated part of the process. The next steps are doing hand-painted details to make the damage texture more interesting.

8. Create a new layer. Here, use the airbrush, paintbrush, and any other painting tools to darken structural details on the hull. You'll want to emphasize the joints between plates and other textured features because they are still there – just charcoalized – and in darkening the texture, you've reduced the contrast that made those features visible. Do not add any glows here, but if you want to paint in exposed decks and structural

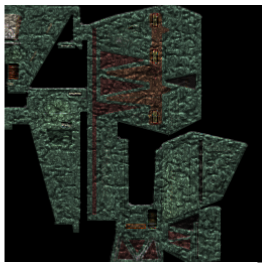


Figure 1.02.12c: Scribbling damage

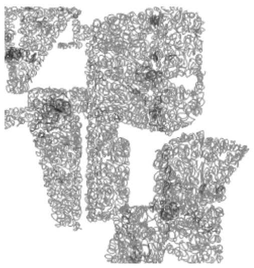


Figure 1.02.12d: Scribbling shown against white, for clarity

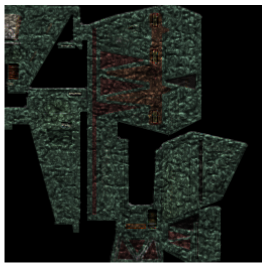


Figure 1.02.12e: After adding a blurred copy of the Scribbled layer

parts below a ruptured hull, knock yourself out. Do remember that you have no idea where the edges of damage will be mapped in, so try not to paint big ruptures that will look strange if only half of them appears.

9. To darken the original texture's warp engines and windows, you may want to go back to the layer you ran Craquelure on and paint them out or darken them there.

10. Above all the existing layers, create another new layer (you'll see why soon). Use the airbrush and paintbrush tools to paint in the yellow-orange-white (or whatever color) glowing bits where the hull, or breaches in the hull, are hot enough to start to melt. Put nothing but these glows in this layer. Keep it up until you are very pleased with the overall effect and sure that the color part of the texture is done. Rework the lower layers if needed.

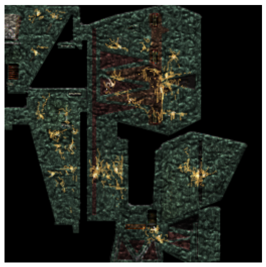


Figure 1.02.12f: Glowing bits painted in their own layer

11. When the color map is complete , create a new layer and fill it with black. Make a duplicate of that glowing layer and place this duplicate above the black layer. Run "Image/Adjust/Desaturate" on the glow layer to turn it greyscale. You may want to adjust its brightness and contrast, too. What you're now looking at is the illumination map for the damage texture. If you want some of the original windows and bright items to continue to glow, copy all or part of the original illumination map above the black layer, below your new glowing layer.

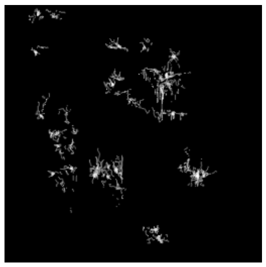


Figure 1.02.12g: The Damage Illumination map

12. Duplicate this image twice. In one duplicate, hide all the layers you're using for the illumination map. In the other, leave the illumination layers (against black) exposed. Flatten both duplicates. One of these will be the damage color map, while the other will be the damage illumination map.
13. You can leave the color map in true color if you want; just remember that you're using a lot more memory by doing so. By all means you should reduce the illumination map to Indexed Color (8 bit or less). Save these two images off using the file naming convention described above "<filename>D.bmp". Put these images in the model directory with the model and its original textures.
14. You're done. Play with your dog for a few minutes, okay? He missed you.

1.02.13 Special Case: Specular Intensity

A specular highlight is the bright highlight you see where a surface is most nearly perpendicular to the direction of the light source that causes the highlight.

3D rendering systems typically let you control the intensity and size of this highlight in one or more ways.

In SFC III, we use a global specular intensity for an entire model (that is, the effect is not controlled by an image map). The specularity of an object is controlled by four values in a .gf file, named to match the model and stored in the model directory (see section 1.03.02.)

Here are the relevant lines from an example .gf file:

```
[Specularity]
Power = 64
Red = 32
Green = 32
Blue = 32
Alpha = 0
```

…and here's what they do:

Power

Legal values: 0-255

Controls the size of the highlights that appear on the model. A value of 0 gives a very broad highlight; a value of 255 gives an extremely small, sharp one. Max users will think of this as the "Glossiness" attribute of a Max material.

Red, Green, and Blue Values

Legal values: 0-255

Control the color and, by extension, the intensity of the specular highlight. Setting all of these to 255 results in a bright white highlight; setting all three to 128 would give a highlight half that bright; setting all three to 0 would, well, give you no highlight at all (black).

Because they are separate values for red, green, and blue, you can also set them to something other than a value of white: set red and blue to 255, and green to 0, and your highlights will be magenta.

All right, that's a pretty lousy example, but you get the idea.

Alpha

Does nothing.

1.02.14 Importing and Exporting Models With 3D Studio Max

At the time of writing, we are still using an unmodified SFC II import/ export plugin for 3D Studio Max. It's available for download from several web sites (including Taldren.com) and can output models for SFC I, II, or III. There are versions of the plugin for use with both 3D Studio Max 3.x and 3DS Max 4.x.

Bear in mind that SFC I can't interpret Max's self-illumination mapping channel, so models that you export with illumination maps will not work in SFC I.

Because the plugin has not been updated – and is compatible with earlier versions of the game – some of its feedback to the user is not really appropriate for SFC III. All you really need to remember is that so far as the export module is concerned, *all* SFC III models are break models. This is because there are no multiple LOD models, just the one.

Install the import/export plugins in your Max plugins directory. Run Max.

To import a .mod file, select "File/Import" and "Starfleet Command" as the format.
Max will ask you if you'd like to replace the entire scene; for our purposes it doesn't matter whether you do or not. Next the plugin will ask if this is a break-up model. Answer "Yes" – remember, as far as the plugin is concerned, all SFC III models are break models. The model, including its hard point and damage point dummy objects, loads into the scene.

To export a .mod file, select "File/Export". This will export the entire scene, not just your current selection, and that's what you want in order to include your hard points and damage points. Make sure that there is nothing in the scene that you do *not* want to export. From the dialog, select "Starfleet Command" as the file format to export. When the plugin asks if you want to "Force Smoothing Across the Entire Mesh", select "No". Next the plugin wants you to select "Yes" for a multiple LOD model, or "No" for a break model; select "No", because you are exporting just one LOD. The file is now saved to disc.

Important Detail:
in SFC III, the model name and its directory name must be exactly the same, like a directory named "Pouting_Pangolin" that contains a model named "Pouting_Pangolin.mod" and a break model version, named "Pouting_Pangolin_brk.mod". See section 1.03.02 for file naming guidelines.

1.03 User Interface Graphics, UI Hard Point Layouts, and Text

1.03.01 UI Graphics

Another new feature in SFC III is your ability to add your ship's picture and hard point layouts to the user interface for the game. It's not possible to change what in-game panels look like, but it is possible to change the images of your new ship that appear within those panels.

We have done this by using external graphics files for the ship images. The game's user interface graphics are compiled into a file called sprites. q3 – for several reasons, some legal and some practical, we can't give you a way to edit these. But those images that show a selected ship are now saved out as .bmp images that are stored in the same directory as your ship model and its textures.

Ship images appear in three players within the user interface. They show up as "schematic" images in the Tactical UI (the "Your Ship" and "Target Ship" panels), and in the Vessel Library and Ship Config screens there are both larger schematic images and a "Beauty Shot" image of the ship.

1.03.02 Reference UI Files, and What You Should Do With Them

You can look in any of the default model directories for examples of these. One thing to note is that in SFC III, the model directory name, model name, text files, and UI images all have to use the same "root" name. An example would be:

Folder name: Dispeptic_Gorilla
- Model name: Dispeptic_Gorilla.mod
- Break model name: Dispeptic_Gorilla_brk.mod
- GF file name: Dispeptic_Gorilla.gf
- Text file name: Dispeptic_Gorilla.txt
- Tactical Schematic image: Dispeptic_GorillaTI.bmp
- Beauty Shot image: Dispeptic_GorillaBS.bmp
- Vessel Library schematic image: Dispeptic_GorillaVL.bmp

In your game's directory, in assets/models, there is now a folder called "modinfo". Within that directory are a set of backdrop images and examples that will help you make UI images that will fit seamlessly into the game's user interface, in the exact same way that we artists at Taldren have done with the game's default ships models.

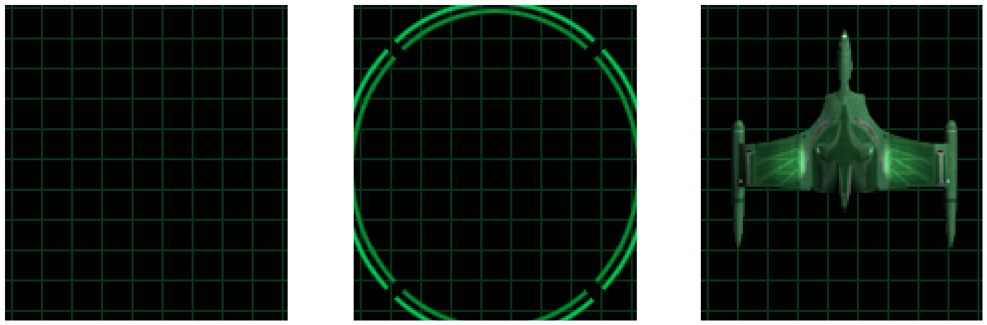


Figure 1.03.02a: The two reference images from your assets/models/modinfo filder and a finished tactical interface image.

The Tactical schematic image uses TIPicBack.bmp as its backdrop. In Max, you can use this as the backdrop image for a top-down rendering of your ship, with the nose pointed up. Another image ("Shieldpic-TIReferenceOnly.bmp") is there for your convenience – this shows how the panel's shield arcs intersect the image area. You can set up your scene using this image as a backdrop, positioning and scaling the ship model to fit within the shield arcs, then switch to the TIPicBack.bmp backdrop for your final rendering. The image size is 141 by 157 pixels.

A similar set of backdrop images is included for the Vessel Library and Config Ship screens. "Shieldpic-ReferenceOnly.bmp" can be used to set up your scene, with a top-down rendering of your ship (nose facing right this time!); then switch to use "VLPicBack.bmp" as the backdrop for your final rendering at 598 by 365 pixels.

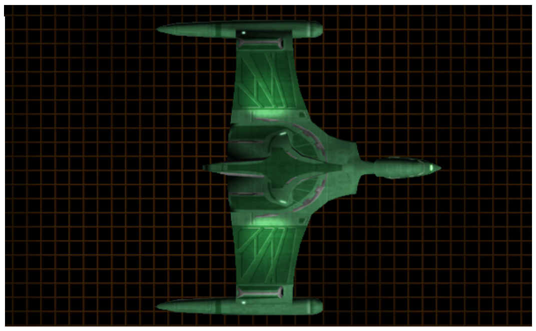


Figure 1.03.02b: A matching UI image for the Vessel Library and Ship Configuration screens.

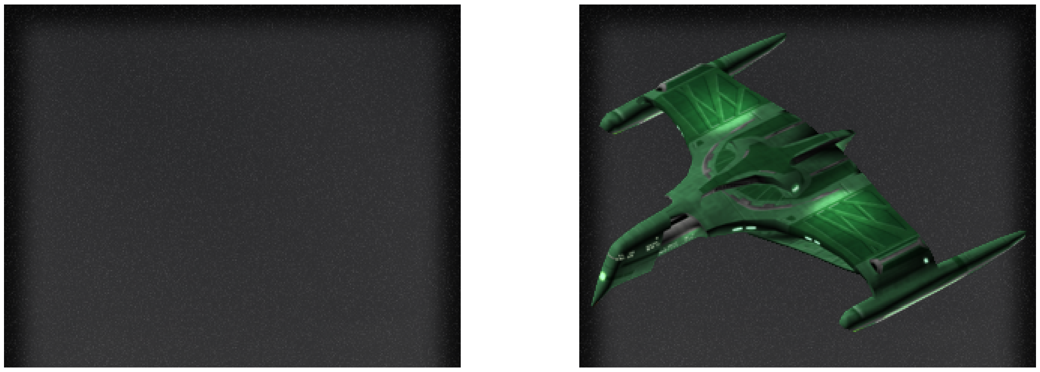


Figure 1.03.02c: Finally, the Beauty Shot reference backdrop and a finished Beauty Shot.

Also on the Vessel Library and Ship Config screen is a "Beauty Shot" of your ship. We do these as ¾ angle renderings, but you can really do whatever you like here. The backdrop image is "BeautyShotBack.bmp", and the image size is 304 by 241 pixels.

1.03.03 UI Hard Point Layouts

Once you've created the UI images of your ship, you'll want to tell the game where the UI should put its hard points. This is done in the text file "<shipname>.gf". You can refer to our existing ships for examples. This is the same text file where you set the specular intensity values for the model.

There is a list of X,Y (horizontal, vertical) coordinates for each hard point. The hard points are referenced by name, and you must be careful to use the exact same names as we've used.

In the assets/models/modinfo directory you will find templates to help you with hard point placement.

"VLPicHardPoints.psd" is the template for the Vessel Library and Ship Config screens. This is a layered Photoshop file in which each hard point type is stored in a separate layer. Primary Weapons (1-11), Heavy Weapons (12-25), Power (engines), Bridge Systems, and Hull Systems are all there in their own layers.

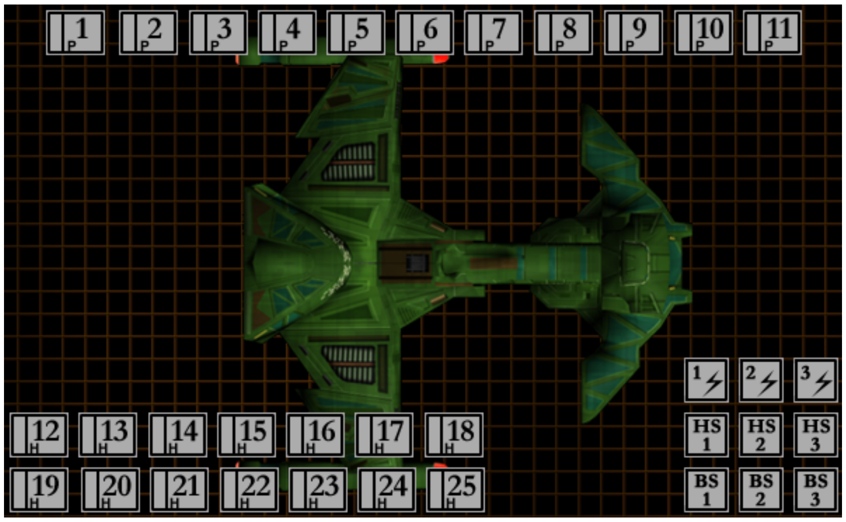
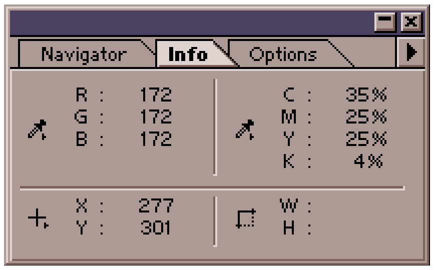


Figure 1.03.03a: We've imported a new VL image and are ready to position its hard points using the Photoshop template.

To use this template, use Image/Duplicate to create a new version of it. Copy your entire Vessel Library image (the one you've rendered with your new model) and paste it into this image right above the "Background" layer.

Now use the hard point layers to move the hard points you need into place, right where you want them to be placed over your ship image. Note that most ships use far fewer than the maximum number of weapons hard points. You can delete any hard points you don't want to use to get the clutter out of the template.

Once they're all in position you will need to get the x,y coordinates of the upper left corner of each hard point. Zoom in and use the rectangular selection tool to hover your mouse right over the upper left corner. In the Info panel you will see the X and Y coordinates for this pixel at the lower left. Those are the values you need to enter in the .gf file.



```
R :  172        C :  35%
G :  172        M :  25%
B :  172        Y :  25%
                K :   4%

X :  277        W :
Y :  301        H :
```

Figure 1.03.03b: At the lower left of Photoshop's Info panel, you can see the X, Y coordinates of your mouse pointer in the image.

You find the X,Y coordinates for the tactical interface panel in the same way, using "TIPicHardPoints.psd" for your template. In the Tactical panel you will only place Weapons and Power hard points.

Here is a sample of the Hard Points section of a .gf file:

```
[HardPoints]
VLBridge1 = 297,141
VLBridge2 = 297,198
VLHull1 = 203,136
VLHull2 = 203,199
VLPower1 = 169,167
VLPower2 = 133,167
VLPower3 = 85,167
VLWeapon1 = 293,21
VLWeapon2 = 293,317
VLWeapon3 = 410,224
VLWeapon12 = 293,54
VLWeapon13 = 293,284
VLWeapon14 = 504,169

Impulse = 72,102
Warp = 48,102
Weapon1 = 15,62
Weapon2 = 105,62
Weapon3 = 81,23
Weapon12 = 36,62
Weapon13 = 84,62
Weapon14 = 60,9
```

Use this same format to enter the X,Y coordinates of your hard point icons, where "VL…" refers to the Vessel Library coordinates, and the second section refers to the panels in the tactical interface.

1.03.04 Ship Name and Description in the Game's User Interface

The ship name (as it appears in the game) is not controlled by the files in the model directory. The final part of this manual is concerned with adding a ship, including its loadout information, to the game – and it's there that you'll see how to control the name of your new hull.

However, in the Ship Configuration and Vessel Library screens there is a scrolling text window where a description of the hull appears. The text that is used in that window is in a text file in the model directory called <shipname>.txt. Have a look at a few.

For the ships that we've included in SFC III, the text descriptions never break character – they always sound like the way those hulls might be described in an actual Star Trek library. There's no reason, though, why you can't include a credit to yourself here, the address of your web site, or a little story about your pet hamster.

If you are kitbashing someone else's model it would be a sign of respect to leave their credit unchanged (or add it, if they didn't include one) and then give yourself credit for whatever retexturing or other modifications you made. This text file could be a chronology of what's been done with a particular well-traveled model.

1.04 Converting SFC I and SFC II Models for SFC III

As we've already seen, the model geometry from earlier models still works in SFC III – but because of some fundamental changes to the engine you will need to modify existing ships before they work well in the game. You should read and understand everything in the previous sections of this manual. Here's a short list of the steps we think you'll need to take in order to make your ships work well in the new game:

1. Delete the LOD2 and LOD3 versions of the ship. (Because of Dynamic LOD, these will all appear together at the same time.)
2. Correct any modeling errors in the ship, including uncapped holes, duplicated vertices or polygons, and intersecting polygons. See sections 1.02.01 – 1.02.07.
3. Make sure that the model's textures are a power of two in each dimension (usually 256 x 256, 512 x 512, or 1024 x 1024 pixels, but could also be 256 x 128, 128 x 1024, etc.).
4. If any parts of your textures are mirrored on adjacent polygons, create duplicates of those mirrored materials and assign one side of the ship to the second material. See section 1.02.09.
5. You will probably want to create a new break model after the materials have been changed; but whether you do or not, number the break chunks and their pivot points so that 1-4 are used on pieces that will look natural if they blow off first. Note that a geometry check is in order for old break models, too. See section 1.02.07.
6. If you're converting an SFC I ship model, you should add illumination maps. See section 1.02.11.
7. Create new Damage textures and Damage illumination maps. See section 1.02.12.
8. Check all hard point names. Hard Points 1-11 will be used only for Primary Weapons, while 12-25 will be used for Heavy weapons. See section 1.02.05. These hard points will correspond to specfile data you enter for the ship when you add it to the game (See Part Two).
9. Create UI backdrop images for the tactical UI and Vessel Library, as well as a Beauty Shot image for the Vessel Library. See section 1.03.02.
10. Add hard point schematic coordinates for the UI panels, in the ship's .gf file. See section 1.03.03.
11. Add Specular Intensity values for the ship in its .gf file. See section 1.02.13.
12. Add a hull description in the ship's text file. See section 1.03.04.
13. Add the ship hull's tactical data as described in Part Two of this manual, to make it appear in the game.

1.05 In-Game Models That Aren't Ships

1.05.01 Planets

Planets are just spherical 3D models with textures applied to them. Adding a planet to the game is pretty much like adding a ship, but the simplest way is to pick a planet you don't like, paint it over, and replace it just by saving your textures over the original files (using exactly the same names).

Planets generally get picked at random, but specific missions may load in a particular planet.

In campaign games the homeworld planets will be used in the hexes where homeworlds are supposed to be, so feel free to put a gigantic Pepsi® banner on Earth, if you like. Someone will eventually anyway.

The planets we've created for SFC III (and SFC II) have a painted dark side. This has two advantages: the texture has to wrap seamlessly at the sides, and it's a lot quicker to make that happen if the sides are black; and our painted dark sides tend to look better than the shading the game engine provides on big spheres.

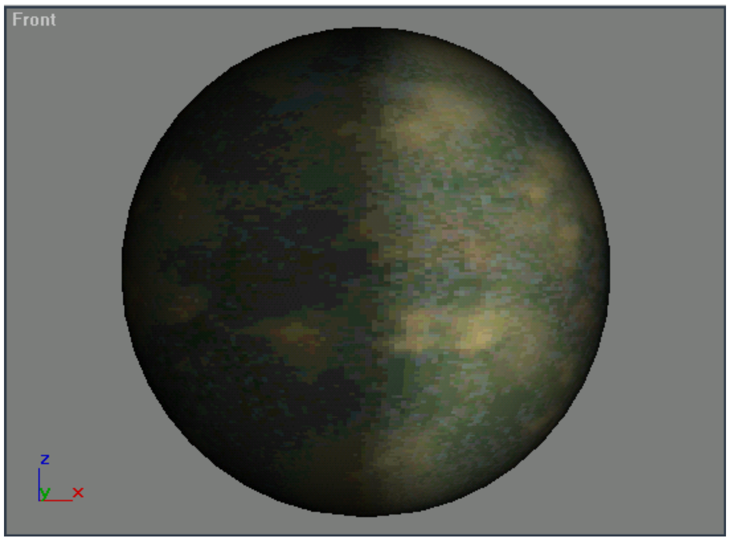


Figure 1.05.01a: This shows the correct orientation (in Max's Front view) for a planet. When exported this way (light side to the right), the game will turn the planet's bright side to face the light source.

The game rotates planets so that their light sides face the light source in tactical; to do that, it assumes that the .mod file of the planet is facing a particular way.

So when you create a planet in Max, before you export it you should double-check its orientation.

In the Front View, the light side of the planet should be facing to the right.

Because planets can be targeted in tactical you will need to create a tactical UI image for your planet – but they don't show up in the Vessel Library, so you won't need one there.

1.05.02 Asteroids

Asteroids are like space backdrops in that you can only replace them – you can't add new ones. In assets/models there are a limited number of Asteroid directories (1-5, at the time of writing) and these are the only ones the game knows to use.

Asteroids show up in any campaign hex where there is asteroid terrain, or in skirmishes if that terrain is selected.

Because asteroids can be targeted in tactical you will need to create a tactical UI image for your asteroid – but they don't show up in the Vessel Library, so you won't need one there.

1.05.03 Space Backdrops

Space backdrops are weird and wonderful things. They use fairly simple geometry with space textures on them. Black areas of the texture become transparent, and the space backdrop object is far beyond the boundaries of the space "arena" of the game – so you'll never bump into one.

If you import one of these into Max you'll find that when they're loaded into the game, these objects are rotated. What is the horizontal "equator" of the space backdrop is a vertical center in Max.

To our knowledge, no one's modded these in earlier versions of the game, and we're not sure why. Give it a shot.

The model files are exported just like ships. The only limitation is that – since they can only be replaced – you need to copy them right over an existing backdrop, with the exact same name: just like you would with asteroids.

Space backdrops are picked at random when you go into tactical in the game. While we're working on one we use a batch file to replace every space backdrop with the one we want to test – that way we know we'll be seeing it.

All the space backdrops, and their textures, are stored in assets/models/space. So we copy our new textures into that directory, copy our new backdrop object there as test.mod, and run this batch file:

NOTE! Make a backup copy of the assets/models/space folder first!

```
copy /Y test.mod space00.mod
copy /Y test.mod space01.mod
copy /Y test.mod space02.mod
copy /Y test.mod space03.mod
copy /Y test.mod space04.mod
copy /Y test.mod space05.mod
copy /Y test.mod space06.mod
copy /Y test.mod space07.mod
copy /Y test.mod space08.mod
copy /Y test.mod space09.mod
copy /Y test.mod space10.mod
copy /Y test.mod space11.mod
copy /Y test.mod space12.mod
copy /Y test.mod space13.mod
copy /Y test.mod space14.mod
copy /Y test.mod space15.mod
copy /Y test.mod space16.mod
copy /Y test.mod space17.mod
copy /Y test.mod space18.mod
copy /Y test.mod space19.mod
copy /Y test.mod space20.mod
copy /Y test.mod space21.mod
copy /Y test.mod space22.mod
copy /Y test.mod space23.mod
copy /Y test.mod space24.mod
copy /Y test.mod space25.mod
copy /Y test.mod space26.mod
copy /Y test.mod space27.mod
copy /Y test.mod space28.mod
copy /Y test.mod space29.mod
copy /Y test.mod space30.mod
copy /Y test.mod space31.mod
```

After running this batch file, our new "test.mod" object has replaced every space backdrop in the game. It will appear every time we play – so as we warned, do back up the original ones before you do this!

When you're satisfied with it, restore the original backdrops, and just copy "test.mod" over one of them. The game will always choose randomly between space00.mod and space31.mod.

You can create a lot of effects like this. One that we've tried is to create a completely spherical space backdrop, texture the lower half as an ocean, and the upper half as a sky. For as long as the game's camera is pointing down at the ships, they appear to be floating in water.

PART TWO

ADDING YOUR SHIP TO THE GAME

PART TWO: ADDING YOUR SHIP TO THE GAME

Get notes from Marc

APPENDICES

.gf file syntax, directory/file naming rules