HOMEWORLD2

SHIP PRODUCTION PIPELINE DOCUMENTATION

1 Introduction

This document is intended to provide a comprehensive guide to *Homeworld2's* ship production pipeline. It assumes that the reader has a basic understanding of modeling and animation within Maya and texture creation with Photoshop. The term 'ship' broadly encompasses all textured 3D meshes that are ultimately exported as `.hod` data (see: Exporting), this includes resources (asteroids and containers), megaliths as well as ship units, their turreted weapons and their subsystems. There will be some redundancies in the document, but this is more for the user's convenience, as it will ease the need to flip back and forth in the document for information… Modifications can get confusing, especially when different processes share similar procedures. It's best to describe them within their own context, like semi-self contained documents.

1.1 Software and Plugins

All HW2 art is created with Maya Builder v3 and Photoshop v5.5. Before embarking on ship creation be sure that all necessary Maya plugins and Photoshop macros are installed in their proper locations.

Installing Maya Plugins

Copy all necessary files into you're the plugins folder in your Maya directory.

Installing Photoshop Export Macro

1. Start up Photoshop
2. Go to the Actions window
3. Clear Actions.... (if there are unnecessary actions in the Actions window)
4. Load Actions...
5. Select Relic.atn from %HW2_ROOT%\tools\Photoshop\Actions directory (where %HW2_ROOT% is where Homeworld2 is installed)

1.2 Process

The steps towards completion of a particular ship are as follows:

1. Modeling
2. Joints and Hierarchies
3. Texturing
4. LODs (Level of Detail)
5. Collision Mesh
6. Ship Animation
7. Exporting

Due to the vagaries of ship production, certain steps may preempt others or may require iterative revisiting. For example, for the purposes of stubbing out game assets, **Exporting** can occur anytime after **Modeling** and **Hierarchy/Joint Setup** are correctly completed. Furthermore, depending on the requirements of a particular ship, additional **Hierarchy/Joint Setup** (like Subsystem Hardpoints or Navlights) tasks may come *after* the **Texturing** process when the ultimate function and form of the ship solidifies.

Depending on the art requirements, the most complex ship file can be comprised of the following elements:

Joints	Meshes	Textures	Animation
Root	Ship Mesh	Layered PSDs	Ship Animation
Weapon	Weapon(s)	⇒ Specular	Docking Paths
Navlights	Docking	⇒ Glow	
FX Spawn Points	Shaders	⇒ Label	
Latch Points		⇒ Stripe	
Repair Points		⇒ Team	
		⇒ Tex	

2 Modeling

Although ideally we should strive for maintaining the most detail one can muster for one's ship while modeling, it is crucial to keep certain parameters in mind that will allow the game engine to run as optimally as possible. Meshes should be modeled with the following considerations in mind to maintain this balance between aesthetic requirements and efficiency.

2.1 Poly Budgets

Considering we have 11,000 polygons dedicated to rendering static geometry, we can assume about 30% of this (3,300 polygons) is available for the highest detail of a ship, including goblins. This is because there will usually be only one top-LOD ship visible on-screen at a given time. If there are multiple ships at the same LOD on screen at a given time, they won't be close enough to render at maximum LOD.

This number should be undercut for smaller ships and can be exceeded for large ships. Be careful exceeding this number, however, as it will affect the global LOD of the game and on lower-end systems, the ship may never be rendered at its maximum LOD.

Assuming a base ship poly count of 3300, here are some guidelines of how the polygons should be allocated on ships of different sizes:

Ship class	% base budget	Poly Budget	Base Mesh	Goblins	% Goblins
Fighter	25%	825	660	165	20%
Corvette	50%	1650	825	825	50%
Frigate	100%	3300	1320	1980	60%
Cruiser/Carrier	120%	3960	1346	2614	66%
Mothership*	175%	5776	1964	3812	66%
Subsystems	2%	74	50	24	32%
Recources	9%	300	300	0	0%
Megaliths*	175%	5776	1964	3812	66%

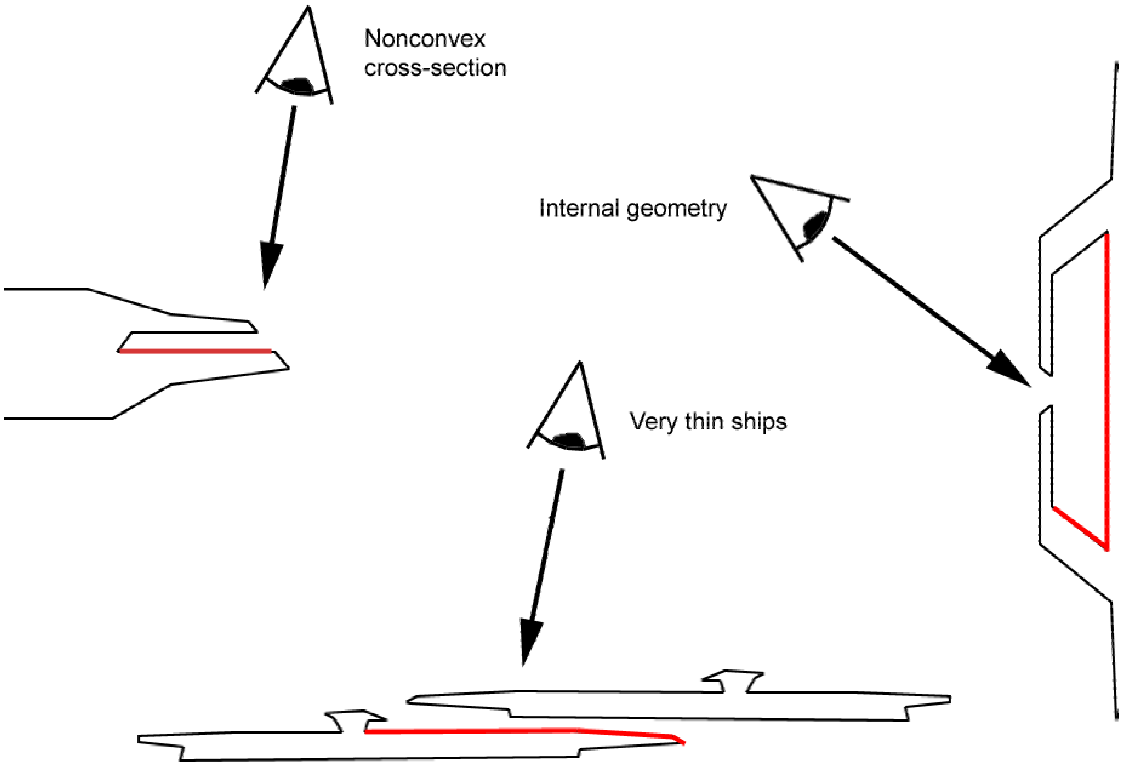
•Motherships and Megaliths might require a special LOD scheme that allows such high poly counts, but do not currently.

For the various LOD levels, we scaled down from the poly counts listed above. For example, if we wanted the polycount of LOD0 without goblins (ie. the base mesh), we just have to look at the Base Mesh value taken from the chart above. There is a minimum number of polygons below which the setup cost is greater than the cost of rendering the polygons. This is about 100, and this will be the minimum number of polygons. LOD1 will be about 1/3 the poly count of LOD0 without goblins (ie. the base mesh). Therefore, we get the following chart:

Ship class	LOD0	LOD0 w/o Goblins	LOD1	LOD2	LOD3
Subsystem	> 100	N/A	N/A	N/A	N/A
Fighter	825	660	220	100	~25
Corvette	1650	825	270	100	~25
Frigate	3300	1320	450	150	N/A
Cruiser/Carrier	3960	1346	470	200	N/A
Mothership	5775	1964	650	500	N/A
Recources	300	300	0	0	N/A
Megaliths	5775	1964	650	500	N/A

2.2 General Topography

Depth buffer precision depicts certain ideal ship topography considerations. In general, ships should be kept roughly convex, not too thin and have limited internal geometry at lower LODs. The reasons for this will be illustrated in the following diagram:



In this diagram, the red thick lines may be drawn in front of the black lines. This usually happens at great camera distances where depth buffer precision is reduced. This can be very unsightly.

Problems with non-convex and internal geometry could be mitigated by manually creating the lower LODs instead of letting the progressive mesh system handle lower LODs.

2.3 Maya Modeling Tools

As previously noted, this document assumes modeling experience in Maya. However, there are certain tools and tricks specific to the process of creating ships for Homeworld2 that must be noted.

X-Ray Toggle

For easy toggling of a mesh to X-Ray mode in order to inspect one's topology in more detail, the Relic Tools shelf includes an X-Ray Toggle button.

Count Polys

In order to maintain optimal poly counts on one's ship, be sure to check your Poly Count status by selecting the ship and clicking on the Count Polys button in the Relic Tools shelf.

2.4 Hierarchies

In order for proper exporting, all ships (including subsystems, resources, and megaliths) require a joint named `Root` under which all objects within the ship file are parented. This joint defines the center-point of a ship.

2.5 LODs and Weapon Effects Collision Meshes

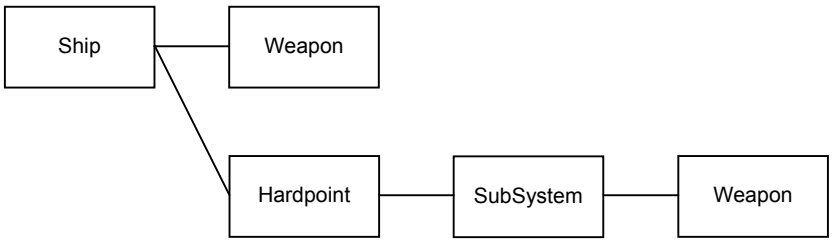
LOD and Collision mesh creation is technically part of the modeling process but is more properly discussed *after* the Joint and Hierarchy application and Texturing, after ship meshes and textures are final (see: LOD (Level of Detail) and Weapon Effects Collision Mesh Pipeline).

3 Weapons

This chapter is meant to be a guide on how to setup a weapon on a ship and on a subsystem. It describes all the rules to follow to ensure you get what you want.

3.1 Ships and SubSystems

The setup of weapons on ships and subsystems is very similar. Is basically follows this pattern:

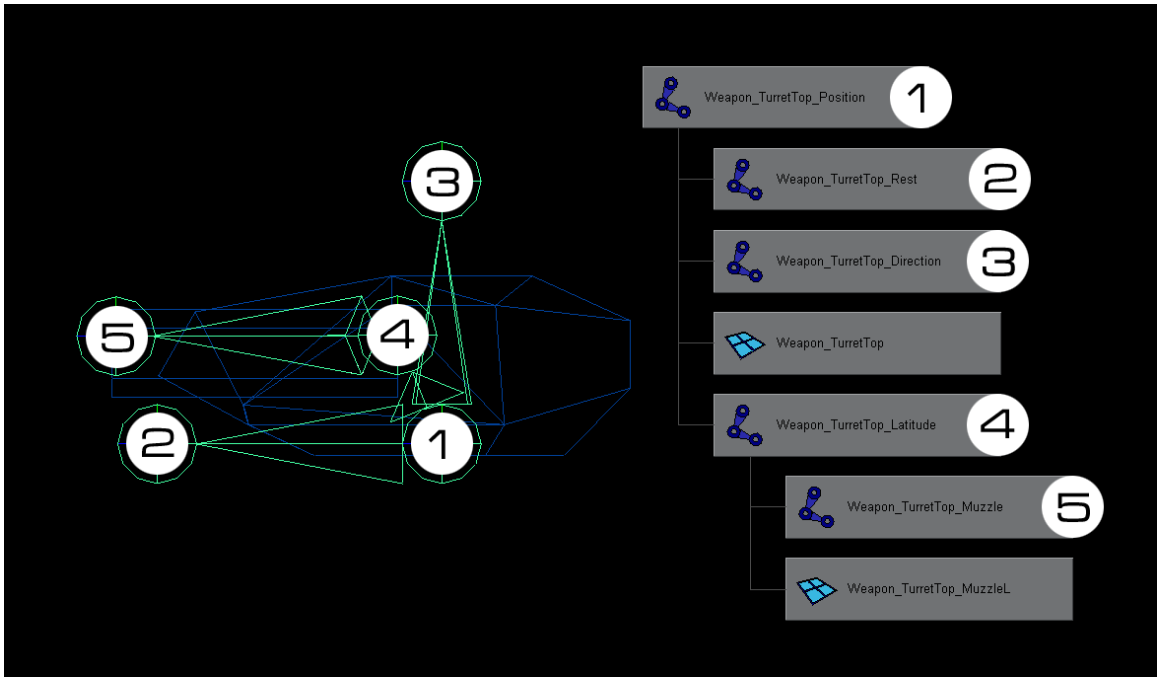


In this figure you can see that weapons can be fitted on ships and on subsystems. The way they are fitted is very similar.

3.2 Joints

Weapons on ships and subsystems need to be modeled and need to have certain joints connected to them. This works the same way as it did before.

Every gun needs a `_Position`, `_Direction`, `_Rest`, and `_Muzzle` joint. In case of a turret, the gun also needs a `_Latitude` joint. The joints need to be named exactly like this, as the game searches for these joints. In the diagram below you can see how one of the guns in the *Hiigaran Assault Corvette* is set up.



1. `_Position` is the general rotation point of the gun
2. `_Direction` is the up vector for a turret
3. `_Rest` is the forward vector for a turret
4. `_Latitude` is the swivel point for a barrel
5. `_Muzzle` is the end of the barrel.

3.3 Two barreled guns

Guns with two barrels are defined by having one master barrel and one slave barrel. Basically you copy the joint hierarchy and add 'Slave' before the _Position, _Rest, etc. So in the example figure above, the name of the slave barrel would be Weapon_TurretTopSlave. The game automatically searches for weapon names with 'Slave' attached to it, so you only need to define them in Maya. Have a look at the Hgn_AssaultCorvette to see how two barreled guns are set up.

3.4 Ships

On ships, multiple weapons can be defined. The weapons defined on ships are part of the ship, so they cannot be targeted or destroyed.

3.5 SubSystems

The mesh for subsystems that are weapons is generally only the gun connected to a root joint.

3.6 Ship and Subsystem Tuning files

Before the guns will work, you'll have to hook up the weapons in the Tuning files (*Data\Ship\Shiptuning.xls* and *Data\SubSystem\SubSystemTuning.xls*). In these files there is a 'Weapons' row. In this weapons-row there are the following fields to fill in per weapon:

Name of the weapon type	Quoted	weaponName	Name of the weapon to use. Look in the Data\Weapon directory for valid weapon names.
JointName	Quoted	jointName	Name of the main joint of a gun. In the example above this would be 'Weapon_TurretTop'

If you fill these in, the game will look for all the joints it needs. The game also looks for slave-guns, so you don't have to define them separately anymore.

3.7 WeaponTuning.xls

The biggest change in the weapon system is that the angles of the gun are removed from ShipTuning.xls. They are now in the WeaponTuning.xls file. This means that, for example, the Hgn_KineticBurstCannon has predefined angles. If you need the same sort of weapon but with different angles, you will have to copy the column in the .xls file to a new column and change the angles. You need to create a directory for the new weapon type in the Data\Weapon directory.

All other variables are the same as they were before. As you can see in the weapontuning.xls file, there are a number of guns specially made for the destroyer. This is a result of the angles being different on different ships.

4 Subsystems

This section is meant to be a step-by-step tutorial on how to create and how to hook up subsystems. It will present the user with a complete path from creating a Maya file to getting it in the game.

4.1 File Structure

This section is meant to explain where all the important files are, and where new files should go. The explanation on how to create new files will be described in the following chapters.

All directories described below are directories under the Homeworld2 main directory (usually something like c:\Projects\Homeworld2\).

DataSrc

The DataSrc directory is where all the source files for the data are stored. This is also where the models for subsystems go. Subsystems are in the directory called **SubSystem**\. In this directory there are a number of directories for each subsystem. Each subsystem needs a unique Subdirectory. Like ships, all files for the Hiigaran are prefixed with Hgn_ and all files for the Vaygr are prefixed with Vgr_.

Files that should be present in a SubSystem* directory should be a **.ma** file (Maya ASCII), and possibly some textures.

4.2 Data

Subsystems

The Data directory is the directory where the game gets all its data. The subdirectories in Data\ resemble the DataSrc directory structure. There is also a **SubSystem** directory. In this directory there are a number of subdirectories again, one for each Subsystem. In each directory there should be a **.hod** file and a **.subs** file.

*Note that the names of the .hod file and the .subs file are **identical** to the subdirectory name. This is important. **If you give a different name to these files, the game will crash**.*

SubSystemTuning.xls

Also in the Data directory is the **SubSystemTuning.xls**. This file is used for tuning all the subsystems in the game. More on how to tune subsystems can be found in the following chapters.

Weapons

There is also a **Weapon** directory in the Data dir. It consists of a number of subdirectories with all the different weapon types described in **.wepn** files, and a **WeaponTuning.xls**. This file is used for tuning the weapons on ships and subsystems. More on this topic will be discussed later.

4.3 Ships

When hooking up subsystems to ships you will need the **ShipTuning.xls** file. This file is in **Data\Ship\ShipTuning.xls**.

4.4 Summary of important files

Tuning SubSytems:	Data\SubSystem\SubSystemTuning.xls
SubSystems (example):	Data\SubSystem\Hgn_FighterProduction\ .hod and .subs
SubSystems (maya example):	DataSrc\SubSystem\Hgn_FighterProduction\ .ma
Tuning Weapons:	Data\Weapon\WeaponTuning.xls
Weapons (example):	Data\Weapon\Hgn_IonCannon\ .wepn
Tuning Ships:	Data\Ship\ShipTuning.xls

4.5 Creating and Exporting the Maya File

Setting up directories

The first thing to do is make a subdirectory for your subsystem in the DataSrc\SubSystem directory. For example, make a directory called Hgn_TestSubSystem. This is the place we will save the Maya file you're about to make for this subsystem.

Also, create a directory in Data\SubSystem\ called Hgn_TestSubSystem.

Modeling your subsystem

Now, start up Maya. You can have a look at how the existing subsystems are modeled by loading the files in DataSrc\SubSystem\. Model the subsystem like you would when modeling a ship.

For now let's create a simple sphere. Every subsystem needs a *Root* node (a joint named 'Root'), just like a ship, so create one and hook it up so that it's actually a root node (set the parent of the sphere to be the Root-joint). Also, don't forget to scale the subsystem according to the ship it's going to be on.

Save the file you just created in the directory DataSrc\SubSystem\Hgn_TestSubSystem as Hgn_TestSubSystem.ma.

4.6 Exporting Your Subsystem

Export the subsystem in the Data\SubSystem\Hgn_TestSubSystem directory. When exporting, select the little box besides the 'export all' menu item. This comes up with a little selection box on how to export the subsystem. Select the 'HOD' file format. You'll notice there is no specific radio-button for subsystem. This is because subsystems export in the same way as ships, so just select 'ship'. Also select the texture format you want to use (default value is good). Then press 'Apply.'

Export the file to the Data\SubSystem\Hgn_TestSubSystem directory, and make sure you name the file Hgn_TestSubSystem.hod. **Remember that the name of the subsystem and the name of the directory need to be exactly the same, or the game will not be able to find the file, and crash**.

That's it! You've just exported the subsystem. Now it's time to hook it up so the game can load it.

4.7 Getting the game to find and load your subsystem

Adding your subsystem to SubSystemTuning.xls

The .Hod file is in place now, but there still is no .subs file in the Data\SubSystem\Hgn_TestSubSystem directory. This file is created by using the Data\SubSystems\SubSystemsTuning.xls file. Open this file in Excel to see how it's done. When Excel is asking if it should enable macros press the 'enable' button.

For now, just copy the 'Hiigaran FighterProduction' column to the right, next to the last subsystem, and call it Hiigaran TestSubSystem. It's important to spell out 'Hiigaran' and to add the space between Hiigaran and TestSubSystem. The excel macros will transform this into Hgn_TestSubSystem automatically.

When you've copied the column, change the Ship Name and Sob Description to 'TestSubSystem'.

If you look further down the column, there is an item that governs Type. You have to fill this one in to let the game know what type of subsystem it is. There are 2 valid types: **System**, and **Weapon**. Type anything else, and the game will crash and let you know that you typed something wrong. For now, just type 'System'.

Exporting your subsystem from SubSystemTuning.xls

Now that you've filled in the column, let's export that column. Select the whole column by pressing on the corresponding button above it labeled by letters. This selects the whole column. Now press one of the 'Create/Update/Export SubSystem Data' buttons. It comes up with a little dialog asking you what you want to send to file. Select 'Send Hiigaran TestSubSystem to file' and press ok. This should have exported it.

Now check if the file was exported correctly by checking if it was created in the Data\SubSystem\Hgn_TestSubSystem\ directory. There should be a Hgn_TestSubSystem.subs file. If it is not there, you have misspelled one of the names somewhere.

That's it! This is all you need to tell the game how to load your subsystem. But you will not see them yet though. Why not? Because you still have to tell the game how to attach the subsystems to the ships. This is done by creating hardpoints on the ship. The next section tells you how to do this.

4.8 Defining Hardpoints on Ships

Maya

Hardpoints need to be defined in somewhat the same way as weapons. They have to be defined in Maya. Open up Maya, and the file of the ship you want to add a hardpoint to. Now create a joint that's linked to the root, and give it the name *Hardpoint_Position*. Now create two extra joints called *Hardpoint_Direction* and *Hardpoint_Rest*, where *Hardpoint_Direction* points straight up (in hardpoint space) and *Hardpoint_Rest* points forward (also in hardpoint space). Then export the ship in the usual way (to a .hod file).

4.9 ShipTuning.xls

Once you've exported the ships you need to define the hardpoints in the shiptuning file in much the same way as weapons. Open up the shiptuning file and search for the column of the ship you just added a hardpoint to.

Now search for the columns called 'Hardpoint'. In these columns you'll find the following members:

Name Of the Hardpoint	Quoted	hardPointName	Name of the hardpoint as it may show up in the game
JointName	Quoted	jointName	Name of the _Position joint without the _Position part (so in our example it would be 'Hardpoint')
Type	Quoted	type	The type of the hardpoint. This can be three things: 'Production', 'System' or 'Weapon'
HealthType	Quoted	healthType	The healthtype of the hardpoint. This can be three things: 'Indestructable', 'Damageable' and 'Destroyable'. Indestructable means that nothing can hit or damage it (nor select it). 'Damageable' means that you can damage it and disable it, but never totally destroy it. 'Destroyable' means you can kill the target, so that the enemy has to rebuild the subsystem from scratch.
DefaultSubSystem	Quoted	defaultSubSystem	Name of the default subsystem. This must be exactly the same as the name of one of the directories in the Data\SubSystem dir.
Family	Quoted	Family	The subsystem family, this drives where the subsystem appears in the taskbar UI, some AI logic and the logic for replacing a subsystem with a new one in the build menu. The valid options: Production, Innate, Sensors, Generic
FittingSubSystem#	Quoted	FittingSubSystem#	Name of the potential subsystems that can be placed on this hardpoint. These must be exactly the same as the name of one of the directories in the Data\SubSystem dir.

So for our subsystem to show up, type 'Hardpoint' in the jointname box, 'Production' in the Type box, 'Indestructable' in the Healthtype box en 'Hgn_TestSubSystem' in the DefaultSubSystem box. Now if you load the ship in the game, the subsystem should show up. That's all there is to it! Below are some specific pointers on how to define the specific variables of Production, System and Weapon subsystems.

4.10 Creating Production SubSystems

Production subsystems are subsystems that enable you to build something. This is linked to the build system. There is nothing you have to do really, just create the subsystem and give it a reasonable name. The buildsystem searches for the name and acts accordingly. If you want the production subsystem to do some extra stuff (like speeding up the build process) then you should

have a look at the Systems Subsystem. The extra attributes that you add to that subsystem can be applied to all subsystems, so you can also add some variables to production or weapon subsystems.

4.11 Creating System SubSystems

General

System Subsystems can have multipliers. Multipliers can modify certain parameters of ships like the speed, repair-rate, buildspeed, etc. If more multipliers are needed but are not available at the moment, just let us know, and we'll add them to the game. These multipliers can be edited in the Data\SubSystem\SubSystemTuning.xls file. Just look for the rows called 'Multipliers', where there are a number of predefined slots of multipliers that look like this:

MultiplierType	Quoted	MultiplierType	The type of variable that is multiplied. Look in the list below on what types are available at the moment.
InfluenceType	Quoted	InfluenceType	Influence area. This tells the system what multipliers to modify, look below for different choices for this variable.
ActivityRelation	Quoted	ActivityRelation	Currently there are two choices: 'None' and 'Linear'. None means there is no relation between the activity and the multiplier. Linear means there is a linear relation between the activity and the multiplier, linking the multiplier to the subsystem's health.
MultiplierHigh		MultiplierHigh	Multiplier for when the activity is 100%, or in case of a 'None'-ActivityRelation, the value of the multiplier
MultiplierLow		MultiplierLow	Multiplier for when the activity is 0%. In case of 'None'-ActivityRelation this variable is ignored
Radius		Radius	Radius for the spheres defined in the InfluenceType. Look below for different choices for the influence type.

Multipliers are multiplied with the variable. This means that if you want a multiplier to be ineffective, you need to set it to 1.0. The multiplier high and low values can be used when the activity relation is linear. The reason there is a high and a low value is more flexibility. For example, let's say you build a speed-enhancer-subsystem. But when the speed enhancer is damaged, it actually has a negative effect on your speed, essentially lowering the speed of your ships below their original speed. In this case you would set the activity relation to linear, the high to 1.2, and the low to 0.8 for example. This causes your ships to speed up when the activity (and therefore health) is 100%, and when the activity is 0% (heavily damaged) the speed is 0.8 times the original speed.

MultiplierType

At the moment the following multipliers are available:

MaxHealth	Maximum health multiplier
Speed	Maximum speed multiplier
BuildSpeed	Buildspeed of ships multiplier

InfluenceType

There are a number of influence types available:

ThisShipOnly	Act only on the ship that this subsystem is deployed on
AllShipsWithinRadius	Act on all ship in the given radius (including the ship this subsystem is deployed on)
OwnShipsWithinRadius	Act on all ships owned by the player within the given radius (including the ship this subsystem is deployed on)
EnemyShipsWithinRadius	Act on all enemy ships within the given radius
AllShipsWithinRadiusExcludingThisShip	Act on all ships within the given radius (excluding the ship this subsystem is deployed on)
OwnShipsWithinRadiusExcludingThisShip	Act on all ships owned by the player within the given radius (excluding the ship this subsystem is deployed on)

4.12 Creating Weapon SubSystems

Creating weapons as subsystems is a bit more complicated, but follows the same pattern as creating a weapon for a ship.

The mesh for a subsystem that is a weapon is basically the weapon itself with maybe a little base attached to it. It needs to have a root and a _Position, _Rest, _Direction, and _Muzzle joint. Also, if the weapon is a turret, it needs a _Latitude joint. The exporting is the same as a general subsystem.

Hooking them up in the Data\SubSystem\SubSystem.xls file is the same as how it works in the Data\Ship\ShipTuning.xls file. Find the 'Weapons' row. The only two things you have to fill in are:

Name of the weapon type	Quoted	weaponName	Name of the weapon used. Look in Data\Weapon* for the names to use in this field
jointName	Quoted	jointName	Name of the _Position joint of the weapon, without the _Position part.

This should hook up the weapons. If you want to create a weapon subsystem, be sure to set the subsystem type to Weapon and be sure to deploy it on a weapon-hardpoint. Slave guns are automatically found and hooked up. Look in **HW2_WeaponsHowTo.doc** for a detailed explanation on how to create and hook up weapons.

4.13 Extra Variables in SubSystems

Self Repair Rate	selfRepairRate	Rate of repair. The given number is the number of health units repaired every second
Collateral Damage	collateralDamage	The damage the ship takes when this subsystem explodes. The given number is the number of health units the ship loses when this subsystem explodes.
Inactive Time After Damage	inactiveTimeAfterDamage	Only used in 'Damageable' Subsystems. This is the time the subsystem remains inactive after being heavily damaged.
Activate Health Factor (0-1)	ActivateHealthPercentage	The amount of health this subsystem is given after it has been inactive for *inactiveTimeAfterDamage* seconds. The given number is multiplied with the maxhealth of the subsystem to set the health.
Build cost in RU	CostToBuild	Total cost of the subsystem in RU's.
Build time in seconds	timeToBuild	The total time to build this subsystem
Is Research system	isResearch	Must be set to 1 if it is a research unit, else it must be set to 0.

4.14 Adding Animation

When subsystems are built, and are deployed on the ship, it's a lot nicer if they don't 'pop' on the ship, but are nicely animated to come out of the ship. This is possible now. All you have to do is animate the subsystem in Maya, and create two animations. One needs to be called '**Deploy**', and one needs to be called '**Recycle**' (they need to be titled *exactly* like this). Obviously, 'Deploy' is played when the subsystem is just built, and being deployed on the ship. The 'Recycle' animation is played when the subsystem is decommissioned/recycled/sold.

Remember that you cannot animate the root node! If you want the whole subsystem to move up or down in the animation, you'll need to create a new joint to facilitate this. Look in the Hgn_Research subsystem on how to implement the animation.

5 Innate Subsystems

5.1 What are Innate Subsystems?

As you may know, *Homeworld2* has support for regular subsystems like ability modules (cloak generators, hyperspace modules, etc.), sensors subsystems, and certain weapons systems (like the Hiigaran Battlecruiser's Ion Turrets). These all use separate, distinct meshes that are considered independent of the hull, for our purposes.

However, **Innate Subsystems** are slightly different. By necessity, these are subsystems that use part of the parent ship's mesh to represent themselves. Examples of Innate Subsystems include things like the Vaygr Battlecruiser's Heavy Fusion Missile Battery, all of the engines for ships in the game that are destroyer sized and up, and resource drop off points on motherships, shipyards, and carriers. Therefore, they always exist on a ship. They cannot be built or destroyed, either.

Making these Innate Subsystems isn't as simple as just making a subsystem mesh and hooking it up to a joint as it is in the case of regular subsystems. The creation process is a bit more convoluted, although the result is the same. This is by necessity due to their nature:

1. Because they use part of the parent ship's mesh to visually 'represent' it, it is invisible.
2. Furthermore, regular subsystems automatically glow green when you mouse over them. Innate subsystems, because they are not visible (as they are 'represented' by the parent ship's mesh), require 'glow meshes' that are not normally visible but when moused over, their shapes glow like a regular subsystem.
3. Simple weapons effect collision meshes are required so that the innate subsystem can get hit and damaged.

5.2 Maya

Many things are done in Maya with the help of various custom Maya plugins. These are designed to enable a developer to add additional content to the base *Homeworld2* ship meshes. In this case, Maya is used in the creation of innate subsystems, specifically their glow meshes and their weapons effect collision meshes. Furthermore, Maya is used in the placement of joints that determine where the innate subsystem gets hooked up.

5.3 ShipTuning.xls, .ship , SubsystemTuning, and .subs Files

ShipTuning.xls is a critical file that gives *Homeworld 2* ships their properties and abilities. In the case of all subsystems, the ability to have them is defined as desired per ship within ShipTuning.xls and then exported to a .ship file. The actual properties of the subsystems themselves are defined within SubsystemTuning.xls and exported to a .subs file. The .xls files are opened in Excel.

5.4 File Extensions and Directories (Folders)

There are a few general things that you must keep in mind regarding file extensions and directories before you begin adding innate subsystems (or any other customization process).

Ships in *Homeworld 2* begin as meshes saved as Maya ASCII files (**.ma** extension). These .ma files are stored in **Homeworld2/Datasrc/Ship/<name of ship>**, along with any associated textures (such as hull textures, badges, LOD textures, etc.). Exported files are exported into a file with a **.hod** extension, stored in **Homeworld2/Data/Ship/<name of ship>**.

With regards to adding innate subsystems to a ship, they must be defined in the appropriate places in **ShipTuning.xls** for the ship in question (found in **Homeworld2/Data/Ship**) and an appropriate **.ship** file (placed in **Homeworld2/Data/Ship/<name of ship>)** must be generated.

Furthermore, subsystem meshes are saved as Maya ASCII files (.ma) as well. These are stored in **Homeworld2/DataSrc/Subsystem/<name of subsystem>**, along with any associated textures. In the case of innate subsystems, a 1x1 pixel .tga is needed in this directory too. Exported files are exported into a file with a **.hod** extension, and stored in **Homeworld2/Data/Subsystem/<name of subsystem>**.

Subsystems have their own version of ShipTuning, and in this case it is not surprisingly called **SubsystemTuning.xls**, which can be found in **Homeworld2/Data/Subsystem**. All subsystem properties, whether for regular and innate subsystems, are found here. Like it's counterpart, SubsystemTuning.xls generates **.subs** files. These are stored in **Homeworld2/Data/Subsystem/<name of subsystem>**.

As you can imagine, quite a few files are affected. We will go into further detail about these files and their directories as we get to them, but to summarize:

Homeworld2/Datasrc/Ship/<name of ship>

- ⇒ **<name of ship>.ma**
- ⇒ **associated textures**

Homeworld2/Data/Ship

- ⇒ **ShipTuning.xls**

Homeworld2/Data/Ship/<name of ship>

- ⇒ **<name of ship>.hod**
- ⇒ **<name of ship>.ship**

Homeworld2/DataSrc/Subsystem/<name of subsystem>

- ⇒ **<name of subsystem>.ma**
- ⇒ **1x1 .tga file (for glow mesh)**

Homeworld2/Data/Subsystem

- ⇒ **SubsystemTuning.xls**

Homeworld2/Data/Subsystem/<name of subsystem>

- ⇒ **<name of ship>.hod**
- ⇒ **<name of ship> .subs**

5.5 How To Add Innate Subsystems

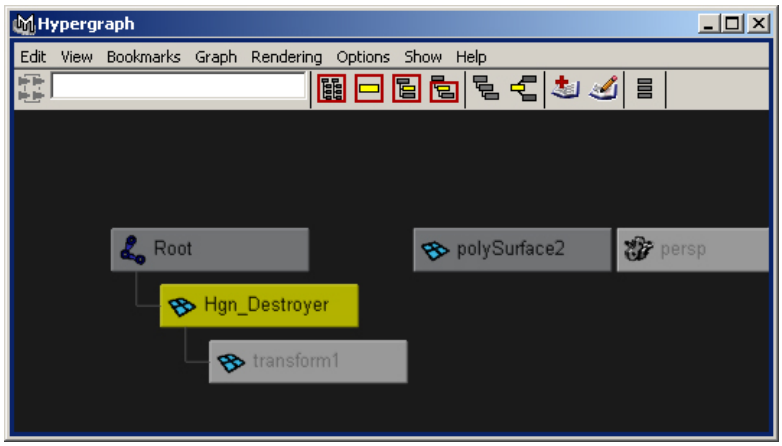
Note: This tutorial assumes that you know how to use Maya and Photoshop. If you need to, please familiarize yourself them before continuing. Furthermore, this tutorial assumes that you have the necessary plugins already installed. Also, you are warned ahead of time that this process is very complex and tedious and not for the faint of heart.

Creating the Glow Mesh

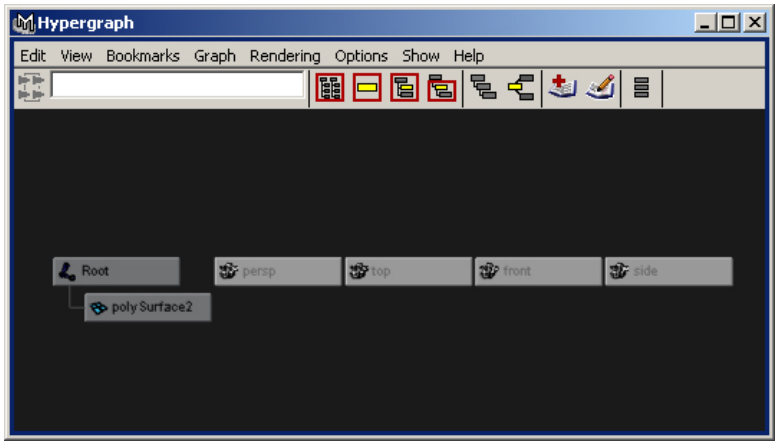
Let's pretend that you're the one who is making the engine of the Hiigaran Destroyer a targetable, damageable, innate subsystem. First of all, we must fire up Maya and load up Hgn_Destroyer. Make a duplicate of this to work on. Call it something other than Hgn_Destroyer - like Hgn_Destroyer_Work or something similar.

1. In the top left corner of the screen, make sure that the drop box displays 'Modeling'. This means that you are in Modeling mode, which is what we want.
2. Examine the engine area of the Destroyer with faces in mind. We will select these faces and extract them for our use in the innate engine subystem. We might have to split up a couple of polygons to get a clean 'cut' so that the engine can be separated cleanly, without weird, uneven edges.
3. Shift-select all the faces (including new ones that you may have made from adding edges) that would make up the rear engine area of the Hiigaran Destroyer.
4. Select Edit Polygons -> Extract.
5. Select Edit Polygons -> Separate. This should split off your selection from the main mesh into a separate object.
6. Make sure that you have object selection turned on (press F8).
7. Select the engine object, and only the engine object, and then focus on its entry in Hypergraph.
8. Drag the object out of the root hierarchy.
9. With the object still selected, go to File -> Export Selection. Select the box beside the entry.
10. Make sure that the filetype is .ma and then click on the 'Export Selection' button.
11. Give a name to this extracted file - Hgn_Des_Engine.ma. Make sure it's saved in **Homeworld2/DataSrc/Subsystem/Hgn_Des_Engine**. If this directory does not exist, make it.
12. Close this current duplicate of the destroyer that you are working on (i.e. Hgn_Destroyer_Work), and load up Hgn_Des_Engine.ma.
13. Go into the Camera Attribute Editor and change the far clip plane camera attribute to a high number like 10000. Your object should now show up properly after zooming in and out.

In game, the target centre of an innate subsystem is where its root joint joins onto the subsystem joint it is assigned on the target ship. Note that in our case, the root joint is far away from where the innate subsystem glow mesh is. We will change this shortly. Examine the Hypergraph. It should look something like this:

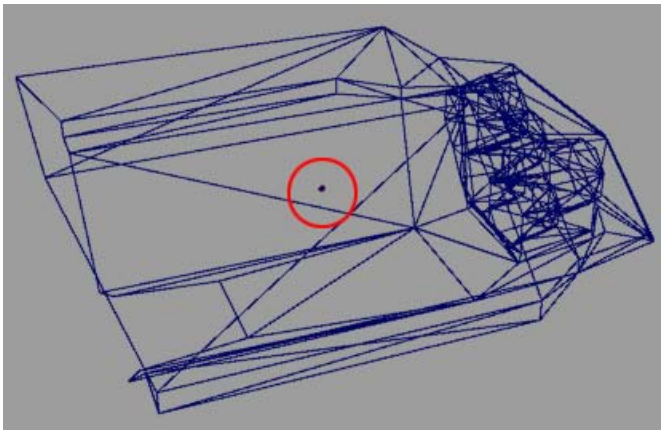


14. Do you see the highlighted Hgn_Destroyer object? Drag it out. Now, drag the object labeled 'polySurface2' in the picture and drag it into Root. Delete the now separated 'Hgn_Destroyer' and its child, 'transform1'. You'll notice that Hypergraph should look like this:



15. 'polysurface2' will be our innate subsystem glow mesh. You can rename it to 'select_Hgn_Des_Engine'.
16. You would also note that the mesh will now look transparent. No matter… Create a new HW2 shader by pressing the 'Create HW2 Material' button (the sphere with HW2 in it, in the Relic Tools Shelf). Open up Hypershade and call the material whatever you like.
17. Open up the new material's Attribute Editor and assign a 1x1 .tga of a single white pixel to it via the button at the right side of the Color slider. This .tga can be named anything you want, but we called it 'white.tga' because we made it a white 1x1 pixel.
18. Go to HW2 Shader Attributes in the Attribute Editor for the same material, and assign it '**innateSS.st'** from the list of shaders.

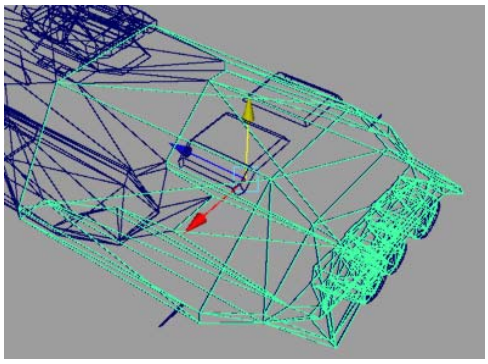
19. Delete all unused shaders from Hypershade. Furthermore, go into Window -> Rendering Editors… -> Multilister… and then from there, Edit -> Delete Unused shaders. This will delete any extraneous shaders that are not used in this mesh.
20. Assign this new material to select_Hgn_Des_Engine. It should now appear as a solid color in Maya.
21. Make sure that select_Hgn_Des_Engine is selected. Activate the Move Tool.
22. Use the command 'Center Pivot' (we've assigned ALT-C as the hotkey for it, although you may have a different assignment) to center it on select_Hgn_Des_Engine.
23. We will now use the command 'Snap to Point' (we've assigned 'v' as the hotkey for it, although you may have a different assignment) to snap Hgn_Des_Engine onto the root node. To do this, make sure select_Hgn_Des_Engine is selected, hold down 'v' (or your own hotkey) and then hold down the middle mouse button at the same time. Drag the pointer over the root joint and Maya will snap the innate subsystem to where the root node is. This is what it the result should look like in wireframe display (the circle indicates the root node:



24. Click on Modify -> Freeze Transformations.
25. Simplify the mesh for the innate subsystem, while retaining as much of the overall profile as possible. Nooks and crannies can be eliminated however, because when they are rendered in game, they will be bad looking bright spots caused by overlapping, redundant faces. Since no edges will be visible, we will aim for as flat of a surface as possible without changing the shape of the engine too much. As usual, manipulating/merging vertices and deleting edges are your best bet towards this. Make sure to watch for triangular, questionable faces on your mesh or else they will not render properly in game.
26. Save your work as **Homeworld2/DataSrc/Subsystem/Hgn_Des_Engine/Hgn_Des_Engine.ma**!!

Hooking Up the Glow Mesh

It's time to 'fit' it onto the ship it's intended for.



1. Make sure you have Hgn_Des_Engine.ma saved. Load up your duplicate of the Hiigaran Destroyer (i.e. Hgn_Destroyer_Work from Section 2.1).
2. Select File -> Import… and pick Hgn_Des_Engine.ma.
3. You'll notice that the_Hgn_Des_Engine will appear in the scene.
4. Use the Move Tool and drag select_Hgn_Des_Engine so that it fits over the engine of the destroyer. **Make a note of the XYZ coordinates in the Channel Box**. These coordinates are what you will use to place the subsystem joint (to position the innate subsystem in game). The picture to the left is an example of what it should look like in Maya in wireframe display (for clarity).
5. Close the duplicate Destroyer mesh.
6. Load up Hgn_Destroyer.ma. We will now add a subsystem hookup joint for the Destroyer engine. Make sure you enable Show -> Joints in the view that you are using.
7. In the top left corner of the screen, make sure that the drop box displays 'Animation'. This means that you are in Animation mode, which is what we want.
8. Click on Skeleton -> Joint Tool. Now click on anywhere in the display near the mesh.
9. Switch to Move Tool and select the joint. Duplicate this joint and then move it negatively along the Z-axis.
10. Duplicate the original again and move it up along the Y-axis.
11. In Hypergraph, join both of these to the original joint.
12. Rename the root of this joint 'Hardpoint_Engine_Position'. Rename the joint placed down the Z-axis 'Hardpoint_Engine_Direction'. Rename the joint placed along the Y-axis as 'Hardpoint_Engine_Rest'.
13. Enter the coordinates that you copied down in Step 4. The point should move to where the center point of the innate subsystem would be in game.
14. Select Edit -> Delete All By Type -> History.
15. Save in **Homeworld2/DataSrc/Ship/Hgn_Destroyer**.
16. Click on the box beside File -> Export All. This will open the 'Export All Options' window.
17. Beside where it says 'File Type,' there should be a drop box. Click on it, and select '**hod**' from the list.
18. Make sure that the '**Ship**' radio button is selected in the 'Export Type' section.
19. Make sure that 'DXT5(rgba)' is selected in 'Texture Options.'

20. Make sure that in the 'Optimization Options' that Triangle List is selected and Merge is selected.
21. Once this is done, click on the 'Export All' button.
22. Select the directory the .hod is going into. In the case of our Hiigaran Destroyer, we will place its **Hgn_Destroyer.hod** in **Homeworld2/Data/Ship/Hgn_Destroyer**. If the directory doesn't exist, you will have to create it.
23. Close the file.

Finishing the Innate Subsystem Mesh – Weapons Effect Collision Mesh

Now that you have your joint placed on the main ship, it's time to finish Hgn_Des_Engine.

1. Reload Hgn_Des_Engine.ma.
2. You might want to scale up select_Hgn_Des_Engine at your discretion. It needs to be slightly bigger than the engine on the parent ship. Use your judgment as to how to go about this. You might also find that you will have to come back and adjust other parts of the mesh to make it fit better.
3. With the select_Hgn_Des_Engine mesh still selected, enter '90' in Rotate X in the Channel Box. This will rotate the subsystem to its proper orientation in the game. It will align itself to the joint placed in Hgn_Destroyer.
4. Duplicate this mesh. This will be your **weapons effect collision mesh**. Rename it to 'CM_Hgn_Des_Engine'. It is highly recommended that this geometry be simplified if possible. This mesh won't be seen at all, and is only used to determine where weapons hit and their effects get spawned.
5. Select Edit -> Delete All By Type -> History.
6. Save.
7. Click on the box beside File -> Export All. This will open the 'Export All Options' window.
8. Beside where it says 'File Type,' there should be a drop box. Click on it, and select '**hod**' from the list.
9. Make sure that the '**Ship**' radio button is selected in the 'Export Type' section.
10. Make sure that 'DXT5(rgba)' is selected in 'Texture Options.'
11. Make sure that in the 'Optimization Options' that Triangle List is selected and Merge is selected.
12. Once this is done, click on the 'Export All' button.
13. Select the directory the .hod is going into. In the case of our Hiigaran Destroyer Engine, we will place its **Hgn_DesEngine.hod** in **Homeworld2/Data/Subsystem/Hgn_Des_Engine**.
14. Close the file.

Adding Values to SubsystemTuning.xls

Your destroyer engine innate subsystem won't work unless it exists as a subsystem as defined by SubsystemTuning.xls. More on SubsystemTuning can be found elsewhere in this document.

1. Load up SubsystemTuning.xls in Excel.
2. It is highly advised to make a duplicate of an engine type as a foundation to work on. Highlight the Hiigaran Carrier Engine column and duplicate it. Rename the new column Hiigaran Des_Engine.
3. Change the Ship Name and Sob Decription to 'Engine'. Under Basic Subsystem Stats, find the row called 'TypeString' and rename it to DestEngine.
4. Click on where it says 'Hiigaran Des_Engine'
5. Click on the 'Create/Update/Export Subsystem Data' button.
6. This will export a .subs file called **Hgn_Des_Engine.subs** into **Homeworld2/Data/Subsystem/Hgn_Des_Engine**.
7. Save and exit.

Adding Values to ShipTuning.xls

Now that your innate subsystem has been defined in SubsystemTuning.xls, you can call them in ShipTuning.xls…

1. Load up ShipTuning.xls in Excel.
2. Go to the column named 'Hiigaran Destroyer'.
3. Scroll down to 'Hardpoints'.
4. Take the first unoccupied spot (usually Hardpoint0) and make sure you enter the following values, ignoring the rest:
    a. Name Of the Hardpoint: **Engine**
    b. JointName: **Hardpoint_Engine**
    c. Type: **System**
    d. Family: **Innate**
    e. Healthtype: **Damageable**
    f. DefaultSubsystem: **Hgn_Des_Engine**

HARDPOINTS					
HardPoint 0			function	StartShipHard	
		Name Of the Hardpoint	Quoted	hardPointName	Engine
		jointName	Quoted	jointName	Hardpoint_Engine
		type	Quoted	type	System
		family	Quoted	family	Innate
		healthType	Quoted	healthType	Damageable
		defaultSubSystem	Quoted	defaultSubSystem	Hgn_Des_Engine

5. The end result should look like the illustration above. By the way, the settings are defined as follows:
    a. Name of the Hardpoint: This is the unique identifier of this hardpoint.
    b. JointName: Name of point in Maya
    c. Type: Can be Weapon, Production, System.
    d. Family: Can be Production, Innate (i.e. engines, resource pads), Generic (hyperspace module, cloak generator, research module, etc.), and Sensors.
    e. HealthType: Can be Indestructable, Damageable, Destroyable.
    f. DefaultSubSystem: The name given is usually a subsystem that the ship starts with.
    g. FittingSubSystem (not shown): The names given are subsystems that the ship can build at the player's discretion.
8. Click on where it says 'Hiigaran Destroyer'
9. Click on the 'Create/Update/Export Ship Data' button.
10. This will export a .subs file called **Hgn_Destroyer.ship** into **Homeworld2/Data/Ship/Hgn_Destroyer**.
11. Save and exit.

And that is how you add an Innate Subsystem to a ship. Go ahead and test the placement. You may find that you have to adjust your mesh. If the innate subsystem is mis-aligned, go back and fix it by changing whatever is necessary (i.e. corrections to rotation, and location… you can try freezing the transformations of the select_ mesh too).

Our example showed how to do this for a Hgn_Destroyer and should be taken as a guideline. However, treat subsystems for each ship as a different case. Sometimes you will find that you will only need to build a simple box shape to cover sections of the hull (as in the Hiigaran Shipyard resource pad). It all depends on the case and the ship in question. Use your judgment.

6 Capture & Repair Points and Navlights

Capture points, repair points, and navlights all have one thing in common - they are defined through the use of joints in Maya. Thus, we will cover the process for adding each one of these to a ship in this document. The rest of this section will describe what's needed first in order to begin.

6.1 Maya

Many things were done in Maya with the help of various custom Maya plugins. These are designed to enable a developer to add additional content to the base *Homeworld2* ship meshes.

6.2 Shiptuning.xls and .ship Files

Shiptuning.xls is a critical file that gives *Homeworld2* ships their properties and abilities. In the case of capture and repair points, the ability only needed to be enabled or disabled as desired per ship within Shiptuning.xls and then exported to a .ship. The .xls files are opened in Excel.

6.3 File Extensions and Directories (Folders)

There are a few general things that you must keep in mind regarding file extensions and directories before you begin the process of adding capture and repair points and navlights (or any other customization process).

Ships in *Homeworld2* begin as meshes saved as Maya ASCII files (**.ma** extension). These .ma files are stored in **Homeworld2/Datasrc/Ship/<name of ship>**, along with any associated textures (such as hull textures, badges, LOD textures, etc.). Exported files are exported into a file with a **.hod** extension, stored in **Homeworld2/Data/Ship/<name of ship>**.

With regards to enabling a ship's capture and repair points, activation values must be toggled on or off in the appropriate places in **Shiptuning.xls** (found in **Homeworld2/Data/Ship**) and an appropriate **.ship** file (placed in **Homeworld2/Data/Ship/<name of ship>)** must be generated.

We will go into further detail about these files and their directories as we get to them, but to summarize:

Homeworld2/Datasrc/Ship/<name of ship>

- ⇒ **<name of ship>.ma**
- ⇒ **associated textures**

Homeworld2/Data/Ship/

- ⇒ **Shiptuning.xls**

Homeworld2/Data/Ship/<name of ship>

- ⇒ **<name of ship>.hod**
- ⇒ **<name of ship>.ship**

6.4 Tutorials

Note: These tutorials assume that you have a working knowledge of Maya.

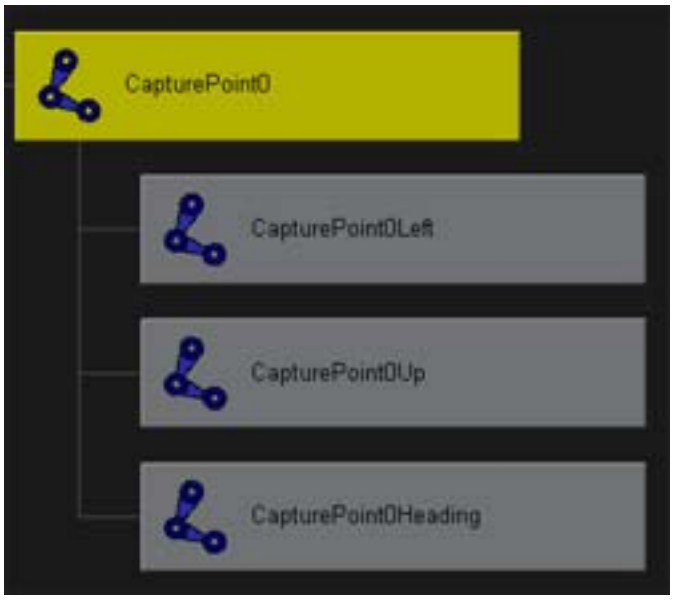
Capture Points

Capture points are quick and easy to make in Maya. Once you have made them, all you have to do is drag and rotate them into position and activate them in Shiptuning.xls. Let's pretend we have a ship called Hgn_Prototype, that doesn't have any capture points attached to it.

Using Maya to Make Capture Points and Exporting Them

1. Once you have Maya loaded up, make sure that under 'Show' in the viewpoint you are using, that 'Joints' is checked off. This will toggle on visibility for all joints in the scene, which includes capture points.
2. In the drop box under 'File' in the main Menu Bar, change the selection to 'Animation.'
3. From the Menu Bar, click on Skeleton -> Joint Tool
4. Click on an empty spot anywhere in the scene. You will now have a joint.
5. Duplicate this joint and drag it a short distance along the X-axis in a positive direction.
6. In Hypergraph, hook up this duplicate joint to be joined under the original one.
7. Duplicate the second joint, and snap it to the location of the first joint. Drag it a short distance along the Y-axis in a positive direction.
8. Duplicate the third joint, and snap it to the location of the first joint. Drag it a short distance along the Z-axis in a positive direction. Your first capture point should now look like the picture.

9. In Hypergraph, find the root of the joint you just made and rename it to 'CapturePoint0'.
10. For the joint that was pulled along the X-axis, rename it to 'CapturePoint0Left'.
11. For the joint that was pulled along the Y-axis, rename it to 'CapturePoint0Up'.
12. For the joint that was pulled along the Z-axis, rename it to 'CapturePoint0Heading'.
13. In Hypergraph, drag 'CapturePoint0' and everything that comes with it, into the 'Root' node.
14. Now, this collection of joints can be called a capture point and should now look like the picture to below in Hypergraph:



15. Close Hypergraph, and drag and rotate the point in Maya to the desired location on your ship.
16. If you want to make another point, either duplicate the point that you have made, or make a new one from scratch. Either way, please insure that the numbers in the names of the various parts of your capture point are increased by one (i.e. CapturePoint1 is the root, and CapturePoint1Left, CapturePoint1Up, and CapturePoint1Heading are the other names for the other parts). Once you are done, place the capture point in another location that is far away from the first capture point.

Exporting Capture Points

1. Make sure you select Edit -> Delete All by Type -> History to delete unnecessary history that may clutter up the file. Save your work.
2. Click on the box beside File -> Export All. This will open the 'Export All Options' window.
3. Beside where it says 'File Type,' there should be a drop box. Click on it, and select '**hod**' from the list.
4. Make sure that the '**Ship**' radio button is selected in the 'Export Type' section.
5. Make sure that 'DXT5(rgba)' is selected in 'Texture Options.'
6. Make sure that in the 'Optimization Options' that Triangle List is selected and Merge is selected.
7. Once this is done, click on the 'Export All' button.
8. Select the directory the .hod is going into. In the case of our hypothetical ship, we will place its .hod in 'Homeworld2/Data/Ship/<name of ship>'. Thus, we will export our Hgn_Prototype.hod file into 'Homeworld2/Data/Ship/Hgn_Prototype'.

Enabling Capture in Shiptuning.xls

The Shiptuning.xls file can be found in the directory 'Homeworld2/Data/Ship'. You will require Excel and enable macros to view it. When you do load up Shiptuning.xls, you will find columns that display the various stats for all of the ships in the game, playable and non-playable. Let's pretend that you've already added a column for a hypothetical ship that you wanted to add capture points for. We'll call it Hgn_Prototype.

1. Find the entry column for Hgn_Prototype. It should be in the Hiigaran side and labeled 'Hiigaran Prototype.'
2. To look at the stats, you might have to expand some rows. Once you have done this, scroll down the column until you reach the section titled 'Can Be Captured'.
3. Find where you have to enter the value for 'Quoted' and enter 'CanBeCaptured'.
4. Below that (in 'time to capture'), enter the base amount of time in seconds that it takes for this ship to be captured.
5. Finally, fiddle with the 'slow down factor' if desired. This will make the ship slow down by a certain amount when ships are trying to capture it.
6. Click on the 'Create/Update/Export Ship Data' button. This will generate a .ship file in Homeworld2/Data/Ship/Hgn_Prototype. Remember, if this was another ship, replace Hgn_Prototype with the name of the ship in question.
7. Save your work.

Voila! Capture points have been enabled in your ship!

Repair Points

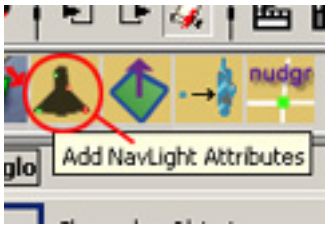
These are done in exactly the same way as capture points. In the case of repair points however, simply replace the name CapturePointX (where X being the number of the point, starting at 0) with RepairPointX. The exporting in Maya, and in Shiptuning.xls is also done in exactly the same way, except in Shiptuning.xls, look for the rows that affect 'Can Be Repaired' and enter 'CanBeRepaired' in the ship's column in the row that says 'Quoted'.

Adding Navlights in Maya

Using Maya

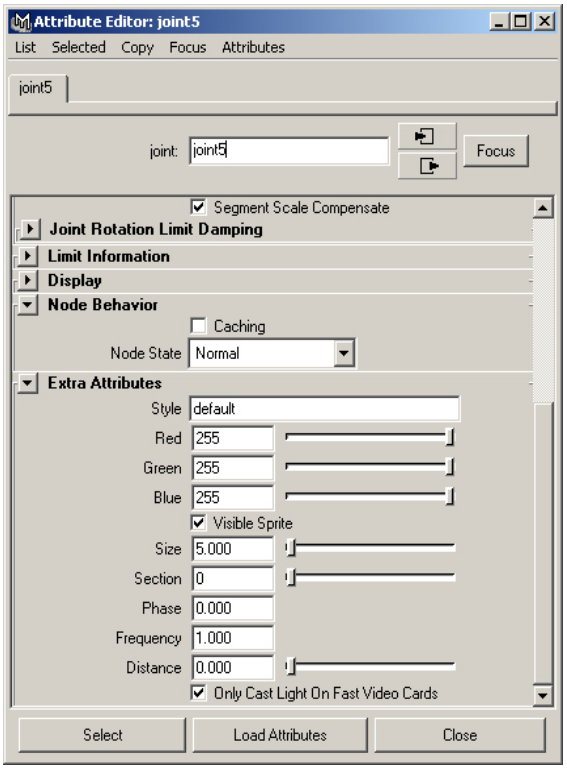
Again, we will use Maya, and a hypothetical ship called Hgn_Prototype.

1. Once you have Maya loaded up, make sure that under 'Show' in the viewpoint you are using, that 'Joints' is checked off. This will toggle on visibility for all joints in the scene, which includes navlights.
2. If you haven't done so already, in the drop box under 'File' in the main Menu Bar, change the selection to 'Animation.'
3. From the Menu Bar, click on Skeleton -> Joint Tool.
4. Click on an empty spot anywhere in the scene. You will now have a new joint. Navlights need only one joint, so we can now call it a navlight point.
5. Name the point 'Navlight1' or anything that has 'Navlight' in the prefix.
6. Click on the Move Tool and move the navlight point to where you want to place it on Hgn_Prototype.



7. With the navlight point still highlighted, click on 'Add Navlight Attributes' in the Relic Tools Shelf.
8. Now, when you call up the Attribute Editor for the object, you will be able to open up 'Extra Attributes' and get a list of relevant navlight specific things that you can adjust and tweak to your liking. Go on to the section about exporting navlights if you wish.
9.

The Attribute Editor – Navlights



This is what the Attribute Editor should look like when you have 'Extra Attributes' expanded.

Notice that you can change the RGB values. You can also toggle the visibility of the 'navlight' sprite. 'Size' adjusts the visual size of the navlight in game. Remember that this affects the way the light is scaled over different distances, however.

'Phase' is the offset in seconds at which the blinking of a navlight plays. 'Frequency' controls the frequency at which the navlight blinks.

'Distance' is used to determine how far out light is cast from the navlight. This is useful if you are making lights to cast a glow on ships coming in and out of another ship's docking bay/hangar. Be sure to turn off the navlight's visibility if this is the case. Furthermore, a maximum of 6 of these are allowed on a single ship. However, these should be used sparingly as 6 light casting navlights will cause the game to slow down considerably. Also, whenever an explosion takes place, the dimmest of these light casting navlights disappears, as an explosion counts for a light casting object.

Once you have finished your alterations, click on 'Close'.

Exporting Navlights in Maya

Exporting navlights in Maya follows exactly the same procedure as exporting capture points.

Exporting Navlights in Shiptuning.xls

There are no navlight properties for ships to adjust in Shiptuning.xls.

Summary

Capture Points

1. Using the Joint Tool found in 'Skeleton' in the Main Menu bar, create a single joint.
2. Duplicate joints as necessary to form a capture point.
3. Name the parts of the joint along the CapturePointX convention (where X is the number of the point).
4. Place point as needed.
5. Duplicate and place point as needed.
6. Export a .hod file into 'Homeworld2/Data/Ship/<name of ship>'.
7. Enable capture property values in Shiptuning.xls.
8. Export a .ship file into 'Homeworld2/Data/Ship/<name of ship>'.

Repair Points

Exactly the same as capture points, except use RepairPointX naming convention instead of CapturePointX.

Navlights

1. Using the Joint Tool found in 'Skeleton' in the Main Menu bar, create a single joint.
2. Place as necessary and name it something with 'Navlight' in the prefix.
3. Add navlight attributes to it and tweak to desired values.
4. Duplicate if needed.
5. Export a .hod file into '**Homeworld2/Data/Ship/<name of ship>**'.

7 FX

Information on the use of the **Homeworld 2 FX Tool** can be found in the document HW2_FX_Tool_Documentation.doc.

8 Texturing

This section outlines the procedures for texturing a ship in Maya for use in HW2. Once a texture is complete, the .psd file will need to be collapsed into a specific format before final export into the game. In order to preserve in-game rendering flexibility (specularity, transparency, etc.) artists must collapse their textures into an intermediate data format. This format is then converted into the final data format the game engine recognizes. It is in this part of the process that Goblins used to enhance surface mesh detail are created, textured and placed. While technically a modeling task, they are more appropriately discussed as part of the texturing process.

8.1 Installing Photoshop Export Macro

1. Start Photoshop.
2. Go to the Actions window.
3. Clear Actions (if there are unnecessary actions in the Actions window).
4. Load Actions.
5. Select Relic.atn from %HW2_ROOT%\tools\Photoshop\Actions directory (where %HW2_ROOT% is where Homeworld2 is installed)

8.2 Texture Budgets

The buffers (depth, shadow, front, back, etc) at 800x600 will take around 16megs. HW2 also used seven megs for the effects, one meg for the UI, four for miscellaneous level textures (debris, asteroids, etc), and four for level specific megaliths for a total of 16 megs. Our two fleets took 16megs each as well. Polygon data was not taken into account as it was assumed to be streamed over the AGP bus, it was around seven megs for each fleet. Even though we use significantly more memory than would fit on a 32meg video card HW2 still ran well on min spec systems thanks to intelligent video hardware memory management and use of AGP memory. Often, even though a ship may have many megabytes of textures on it, only a little bit of that was used in a given scene as the ship may only be a couple pixels big. If your mod is targeting a higher system spec than 32meg then you can afford to increase the texture budget for your fleets.

In order to save texture detail for the largest ships we applied less detail to the small ships. Larger ships needed a lot of detail. A small ship you will never get all that close to, so it will only ever take a portion of the screen. However a portion of a large ship may completely fill the screen.

Here are some typical texture sizes for ships in HW2:

Ship	Texture size (MB)
Probe	0.18
Fighter	0.2
Corvette	0.2
Frigate	0.46
Capital Ship	1.5
Mothership	3.4
Megalith	5.5

8.3 Naming Conventions

Naming conventions. Artists should use the convention <texture name>_TEX.psd for textures that have been collapsed and are ready for final processing before exporting in-game models. <texture name>_EXPORT.psd will be used for textures that have been post-processed for final export.

This means that the artist textures the ship as normal, for example, with a texture called `mainUV.psd` containing a multitude of layers.

When the artist is satisfied with the results and is ready to export into the game, they collapse the .psd into a few specifically named layers (described later on in this document), name the file `mainUV_TEX.psd`.

8.4 Assigning Shaders in Maya

In order assign a texture to one's ship it must be attached to a shader and then applied to the mesh.

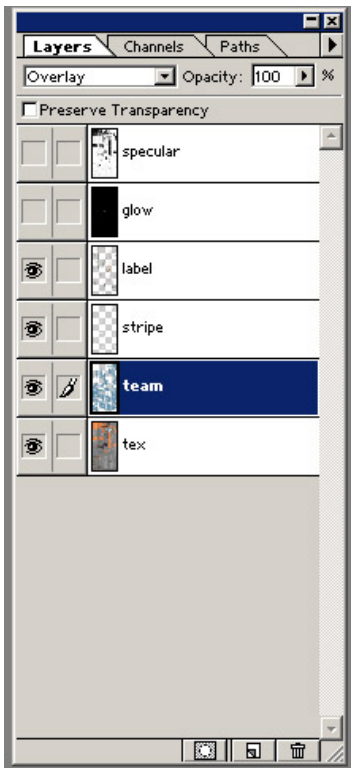
1. Click on the **Create HW2 Material** button in the tool shelf. This will create a new HW2 shader in the Maya's Hypershade window.
2. Double-click on the shader to access its attributes

8.5 Working In Maya

Game-ready textures are created from these files. Even if this game-ready format should change, `mainUV_TEX.psd` should not need to be altered. A script is run in Photoshop and a new texture `mainUV_EXPORT.psd` is created. This produces a texture that may look quite odd to the artist since different masks- specular, glow, base and stripe team colors, etc.- are all packed into opacity and color channels.

The ship's .ma file is then loaded into Maya. The textures on the ship are set to the _EXPORT versions and the ship is exported to a .hod file. Confirm that the export was successful by starting up the game and looking at the ship.

8.6 Layers



The format for the _TEX.psd file should be the following:

On the bottom should be a completely opaque *normal layer* named '**tex**;. This is the base colour of the ship. It should be toned grey anywhere one of the overlay (team / stripe) layers sits over top of it.

Above tex we have two *overlay layers*. These are the two team colors. '**Team**' is the main team color and '**stripe**' is the stripe color. These two layers should be a single constant color with transparency indicating where they go over the base texture.

Above the team color layers we have a *normal layer* called '**label**' with dirt, labels and whatever else goes over the base and team colors.

Next, we have a luminance map, called '**glow**'. This map is all the illuminated elements of ships; engines, bridge lights, hangars, etc. This should be a fully opaque *normal layer* called glow using only grayscale. Black indicates no glow. White indicates full glow.

Finally we have the specular map in a layer named '**specular**'. This map corresponds to the reflective quality on the surface of the ship. This should also be a fully opaque *normal layer* using only grayscale. Black indicates areas without specular reflectivity and white indicates areas with specular reflectivity.

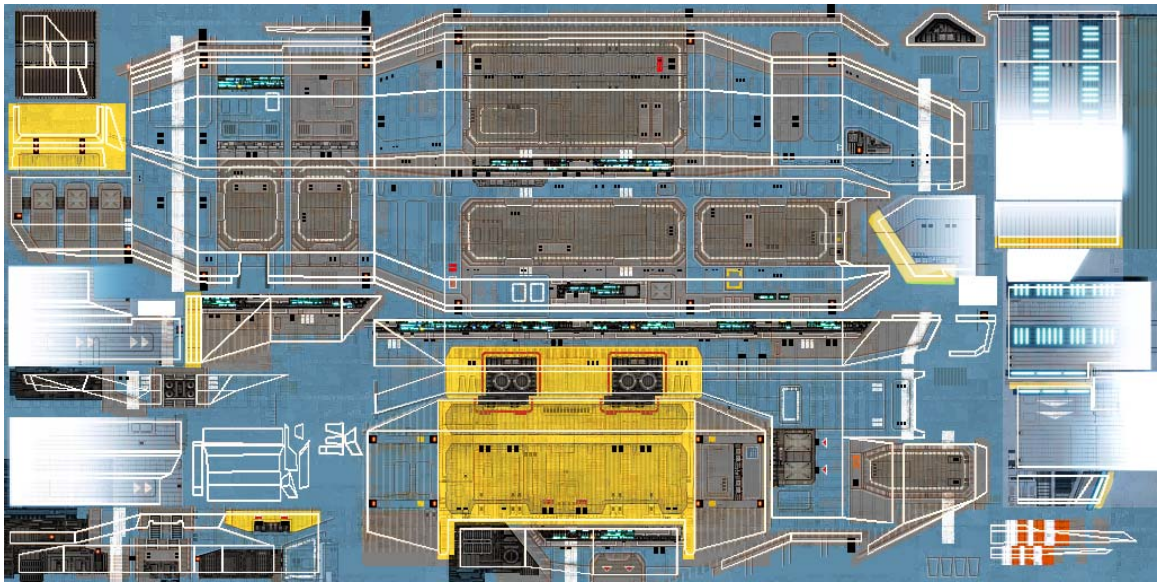
8.7 Texturing Considerations

Our texturing pipeline consists of creating textures in Photoshop using the native .PSD format. This layered image format allows us to create special layers with the base texture that have special meanings:

Layer type	Name in Photoshop	Blending modes	Notes
Base texture layers	TEX	Normal	
Alpha(opacity map)	ALPHA (in Photoshop's Alpha Channel view)	Multiply (for preview in Photoshop)	White is opaque, black is transparent. Actual alpha channel ignored
Decal	LABEL	Normal	
Illumination (glowy) map	GLOW	Screen (for preview in Photoshop)	Greyscale.
Team color layers	TEAM, STRIPE	Overlay	TEAM should be below STRIPE
Helper layers (uvs, goblin placement etc.)	REF	Any	

8.8 U/V coordinate considerations

Examine the following texture, from the Hiigaran Carrier, with UV mapping coordinates drawn on top of it:



Note how there are separate discreet mappings for different parts of the ship. Some considerations to keep in mind while creating mappings:

- Extend the color at the edge of a mapping past the edge of the mapping. This prevents filtering from darkening the edge to black, or whatever other color might be past the mapping.
- Use a minimum of mappings as a polygon edge that borders two mappings will not filter as well as other edges.
- Ensure the other mapping that will meet a given mapping edge extends out at the same color.
- Allow space between mappings, even when scaled down to lower map levels, make sure there is space at least 4 levels down from the top level.

8.9 Badges

Badges are user-defined textures that are placed as decals on the surfaces of ships. Each badge is a 64x64 RGBA texture. Rendering badge areas involves: rendering the ship's base texture (including team colour, stripes, dirt, etc), rendering the badge texture, and rendering lighting.

The following are the steps for placing a single badge on a ship. These steps assume that you are working with the _TEX.PSD versions of the ship texture until export time.

1. Create a _TEX.PSD file for the badge area
2. Place the badge-area texture onto the ship
3. Insert a sample badge into the badge-area texture
4. Reveal ship details using a mask layer
5. Export model and test it in the game

Step 1: Create a _TEX.PSD file for the badge area

1. Determine where the badge would go. Keep in mind that goblins should either be entirely within or entirely outside a badge area.
2. Tessellate the badge area so that it is square. This minimizes the distortion on the badge texture. Ideally, this area should also map into a square or rectangular region in the ship's texture.
3. Create a 64x64 _TEX.PSD file for the badge-area texture.
4. Copy the ship texture layers for the badge area into the _TEX.PSD file and resize it to 64x64.
5. Rotate the badge-area texture (by a multiple of 90 degrees) until it has the same orientation as the badge (i.e. the up direction corresponds to the up direction of the badge texture).
6. Touch up the texture you just copied. Pay particular attention to the pixels on the edges as the texture resizing can alter the values of edge vertices.

Step 2: Place the badge-area texture onto the ship

1. Create a new shader with the badge-area _TEX.PSD file as texture. Name this shader 'Badge0_MAT' for the first badge on the ship. Subsequent badges on the ship should be named 'Badge1_MAT', 'Badge2_MAT' and so on.
2. Set the Shader script name (in the HW2 Shader Attributes) to 'Shaders\badge.st'.
3. Select the faces in the badge area.
4. Assign the badge-area shader to the selected faces.
5. Update the UVs of the selected faces until the badge area smoothly blends in with the rest of the ship.
6. Assign the badge-area shader to any goblins that appear in the badge area.
7. Update the UVs of the goblins.

Step 3: Insert a sample badge into the badge-area texture

1. Create a new layer in the _TEX.PSD file.
2. Insert a sample badge texture into this layer.
3. Save the _TEX.PSD file.
4. Reload texture in Maya and verify that the sample badge is not clipped or grossly distorted.

Step 4: Reveal ship details using a mask layer

An optional mask layer may be used to control the opacity of the badge texture. The mask layer has a few interesting properties:

7. It must be stored in a layer named '$mask',
8. Only the alpha channel of the mask is used, and
9. The colour channels of the mask are ignored.

The mask is useful for alpha-ing out portions of the badge where dirt and panel lines are so that they may show through the badge. This is easy to do if all these details are already stored in a single layer (e.g. the 'label' layer). Then you simply need to copy the 'label' layer and name it '$mask'.

You can also use the mask layer to block off areas where the badge should not appear --just paint those areas with an opaque colour.

The effect of the mask can only be observed in the game.

Step 5: Export model and test it in the game

1. Remove the sample badge layer. This layer is used for testing only.
2. Apply the Relic.atn action to the _TEX.PSD file. The '$mask' layer should still be around after the action is applied.
3. (Optional) Turn-off the '$mask' layer so the texture won't look too goofy in Maya. The '$mask' would export fine with or without this step.
4. Save the export-ready .PSD file out as xxxx_EXPORT.PSD.
5. Update the textures in Maya to use the _EXPORT.PSD versions of the textures.
6. Export the ship from Maya.
7. Verify that the ship looks as expected in the game. Try to use a few instances of the same ships but owned by different players. This would test your ship using different team colours and badges as they are player-specific.

8.10 LOD Texturing

You need to send your triangles in groups of more than 500 to achieve maximum efficiency. So if you have a 600 polygon ship and you break it down into four groups of triangles of 150 each then you'll get about half the triangle throughput vs. one group of 600! As low as 300 you still maintain a reasonable triangle rate, but anything lower than 100 is pointless.

This is assuming the same rendering state. Things get even worse if you have to switch textures or do any changes to the render state, which would presumably be the reason for breaking the ship down from 1x600 to 4x150. In fact, switching textures is so bad nVidia recommends you combine your textures into a single 4096x4096 texture so you have to switch less!

Anyway, 300 is the point at which diminishing returns really start to kick in, but 100 is more of the real limit where anything lower is really not winning any performance at all. I'd guess from looking at the chart dropping from 300 to 150 triangles still takes 75% of the time-even though you're doing half the work. 100 (1/3) takes 62% of the time and 75 (1/4) takes 60% of the time. There is still a small benefit to going below 300, but below 100 is completely useless. Keeping those numbers in mind might be useful for LOD matters.

Have all the mipmaps generated regardless. Imagine you're looking down the side of something with a 512x512 texture (256K). Supposing you can only the first two levels (I imagine you'll be able to see more). By dropping all mipmap levels other than the first two we go from taking 341.33k for everything down to 320k!!! Pretty insignificant win to try not bothering to load the rest of the levels... So if you don't really need to create a custom texture, don't-you've already got the mipmap there for free.

Creating the low-res version typically isn't worth it because the video card is smart enough to choose the lowest level mipmap automatically. The only time it would be a performance win is in times where you are effectively saying ‘I know you should use a high res version of the texture now for sake of appearance, but use this low res version instead’ This will occur rarely, if ever.

8.11 Glow Lighting

A reminder about the way the glow texture works; 0-127 is the same as 0-255 of normal light. If you have a glow value of 127 the texture will be drawn full brightness without any contribution from the game light. Likewise a glow of zero means 100% of the lighting comes from the level lights. When you go above 127 it will cause your base texture to wash out. When setGlareIntensity() is used with a value more than 0 these overbrights seep into the surrounding pixels.

8.12 Goblin Specifications

Goblins are small pieces of detail geometry that are added on top of regular ship geometry to provide greater detail very close-up. This geometry helps to provide a more detailed silhouette for a ship and to provide more lighting detail. While they can have their own texture, they usually borrow the texture from the geometry 'below' them. Here are some considerations to keep in mind when authoring goblins:

- Keep them fairly squat and/or small. Large, tall goblins will be noticeable when they turn on and off, especially if they are on the silhouette of a ship.
- Avoid 'vertical' surfaces on goblins, relative to the polygons they sit on. Since goblins are textured with the same u/v coordinates as the polygons they sit on, vertical surfaces will result in texture 'streaking. This can be avoided by custom-texturing vertical surfaces on certain goblins.
- Don't have the goblins penetrate the surface of the ship too deeply. This may present problems like the internal geometry problem illustrated above.
- Be careful of the overall ship poly count. If you put too many goblins on the ship, they may not all be rendered.
- If you create a goblin with custom-textured vertical surfaces, make sure the texture on the sides matches the texture under the goblin. The goblin may turn on and off when the camera is not facing directly from 'above' the goblin.

8.13 Goblin UVing

Goblins can extract UV coordinates from their base textures and be applied to the mesh in the following fashion:

1. Open the `Goblin Name Tool` 

from the Relic Toolbar for Maya
2. Select the mesh to be Goblinized and assign it the same Shader as your base texture (see: Assigning Shaders in Maya).
3. Position the Goblin
4. Name the Goblin a generic name (this is a legacy procedure and it is not necessary to keep track of the name – it is required only to designate the Goblin mesh as a Goblin to complete this procedure)
5. Select the Goblin and Shift Select the base mesh
6. Click on the `Goblin + Object` tool from the Relic Toolbar for Maya. The Goblin will now assume the UV coordinates from the surface it is sitting upon. 

7. For the exporter to recognize them, Goblins are placed within a Maya layer named `Goblin` or `Goblins`. In turn *all* goblins in a ship file are parented under the `Root` joint in the file.

Note: Don't place goblins such that they straddle material boundaries. This applies for material boundaries that represent different textures or different mappings within a single texture. The Goblin UVing tool will attempt to anchor its UV coordinates across UV patches, this will result in smeared textures. If you really want to place a goblin thusly, it must be manually textured.

8.14 Maya Texturing Tools

NUDGR

The NUDGR is a tool for applying often used basic transforms like rotation and positional tweaks to UV coordinates. Clicking on the icon on the Relic tool shelf will call up the NUDGR panel.

Refresh Textures

During the process of texturing changes can be viewed on one's mesh by clicking on the Refresh Textures button on the tools shelf.

Attribute Editor

Sometimes over the course of duplicating goblins in the process of modeling, ship mesh attributes can be set improperly. This is especially true when mirroring objects. If one's exported ship has inverted goblins, select the goblins in question, call up the Attribute Editor, expand the Rendering Stats section, and uncheck the **Double-Sided** and **Opposite** checkboxes.

8.15 Megalith Texture Reuse File Pathname Considerations

To have two separately exported HOD files in a level use the same texture in memory you need to make sure that the texture paths in Maya point to the exact same **path and file.** Merely pointing to the same filename will not do it; it has to have the same path and filename.

9 Ship and Subsystem Icons

Ship or Subsystem Icons are usually included on a single image file and require a .lua file to indicate the coordinates of a particular icon within the image. The ship Icons, as well as the large SubSystem Icons (for the build menu) and the smaller subsystem icons (for the taskbar) can be located in Homeworld2\data\Ship\Icons folder.

9.1 LUA Files

A typical line of the .lua file for the ship icons (ShipIcons.lua) would look like this:

```
Hgn_AssaultCorvette = {
      LargeIcon = {
            texture = "DATA:\\Ship\\Icons\\icon_ships.mres",
            textureUV_TL = { 0, 704 },
            textureUV_WH = { 150, 64 },
            stretchOnDraw = 1,
      },
}
```

Here it states the ship name, based on the shiptuning.xls file, the icon_ships.mres determines what image file to look at for the necessary game resolution (more about this in a moment). The textureUV_TL indicates the top left pixel of the image (in this case, at location 0 x and 704 y, you will find a picture of a Hiigaran Assault Corvette), and the next line (textureUV_WH) indicates the

width and height of this icon. A typical line near the bottom of this same file (ShipIcons.lua), for Subsystems, would look like this:

```
Icon_Subsystem_CapShip = {
      LargeIcon = {
            texture = "DATA:\\Ship\\Icons\\icon_subsystem.mres",
            textureUV_TL = { 0, 0 },
            textureUV_WH = { 64, 32 },
            stretchOnDraw = 1,
      },
      SmallIcon = {
            texture = "DATA:\\Ship\\Icons\\icon_subs_taskbar.mres",
            textureUV_TL = { 0, 0 },
            textureUV_WH = { 32, 24 },
            stretchOnDraw = 1,
      },
}
```

Each of the lines follows the same rules as the ship icons with the only exception of the necessity to locate two icons, on 2 different files for the same item. In this case above, the Capital Ship subsystem Icon can be found in two files. The first is the larger image used in the build menu, and when singularly selected in the taskbar, and the second is the smaller icon used in the taskbar when the entire ship is selected and you need to see it's entire compliment of subsystems.

9.2 MRES Files

The Icon_ships.mres file (mentioned above), as well as all .mres files determines what image file each game resolution should be using. A typical .mres file looks like this.

```
baseRes = 800
res800 = "DATA:/Ship/Icons/icon_ships_800.dds"
res1024 = "DATA:/Ship/Icons/icon_ships_1024.dds"
res1280 = "DATA:/Ship/Icons/icon_ships_1280.dds"
res1600 = "DATA:/Ship/Icons/icon_ships_1600.dds"
```

baseRes = 800 indicates the Base resolution that the xy calculations for each icon are made by. For example, The Hiigaran Assault Corvette is located at 0 x and 704 y on the image that is used for games played at 800 x 600 (icon_ships_800.dds). All Icons (ship or Subsystem) are based on this base res of 800.

Each of the next four lines indicates what file (and where) is to be used for each of the four supported resolutions. The file format for some files (ship files for example) are dds. Other icon formats will be 32 bit .tga

9.3 Image Files and Their Locations

Ship Icons

- Data\Ship\Icons\ icon_ships_1600.dds
- Data\Ship\Icons\ icon_ships_1280.dds
- Data\Ship\Icons\ icon_ships_1024.dds
- Data\Ship\Icons\ icon_ships_800.dds

Large Subsystem Icons (for the build menu and singularly selected in the taskbar)

- Data\Ship\Icons\ icons_subsystem_1600.dds
- Data\Ship\Icons\ icons_subsystem_1280.dds
- Data\Ship\Icons\ icons_subsystem_1240.dds
- Data\Ship\Icons\ icons_subsystem_800.dds

Small Subsystem Icons (for the taskbar when an entire ship is selected)

- Data\Ship\Icons\ icons_subs_taskbar_1600.dds
- Data\Ship\Icons\ icons_subs_taskbar_800.dds

The 3 mres files for each of these images sets and the single ShipIcons.lua can be found in the same folder (Data\Ship\Icons\)

When selecting more than one ship, the taskbar will display the selected ships on a button background. This button highlights on a mouse-over and flashes when clicked (to select a single ship). The files needed for these images are as follows.

Button Images

- Data\UI\NewUI\Taskbar\ship_button1600.tga
- Data\UI\NewUI\Taskbar\ship_button1280.tga
- Data\UI\NewUI\Taskbar\ship_button1024.tga
- Data\UI\NewUI\Taskbar\ship_button800.tga

The ship_button.mres file is located in the same location (Data\UI\NewUI\Taskbar)

Each Image consists of a normal state, a mouse-over and a click state. Also note, these images are used again in the Launch Menu.

Similarly, when selecting a single ship, the entire compliment of subsystems show up against a series of button background signifying their general function by shape and colour. These files can be located here:

- Data\UI\NewUI\Taskbar\subsystem_button1600.tga
- Data\UI\NewUI\Taskbar\subsystem_button1280.tga
- Data\UI\NewUI\Taskbar\subsystem_button1024.tga
- Data\UI\NewUI\Taskbar\subsystem_button800.tga

The subsystem_button.mres is located in the same location (Data\UI\NewUI\Taskbar)

Each Image consists of a normal state, a mouse-over and a click state (top to bottom) as well as the four types of buttons (left to right, these are: production, sensor, general, and innate)

Finally, most other images for the taskbar, such as the command icons and the taskbar itself can be found in Data\UI\NewUI\Taskbar along with their mres files, and within the New UI directory structure in general.

More information on how UI scripts work and placing mres textures can be found in the New User Interface document, located in the Appendices.

10 LOD (Level of Detail) and Weapon Effects Collision Mesh Pipeline

Many things were done in Maya with the help of various custom Maya plugins. These were designed to enable a developer to add additional content to the base *Homeworld2* ships. In the case of the LOD and weapons effect collision mesh pipeline, part of our custom exporter suite was used to export models that have LODs.

10.1 File Extensions and Directories (Folders)

There are a few general things that you must keep in mind regarding file extensions and directories before you begin the LOD process (or any other customization process).

Ships in *Homeworld2* begin as meshes saved as Maya ASCII files (**.ma** extension). These .ma files are stored in **Homeworld2/Datasrc/Ship/<name of ship>**, along with any associated textures (such as hull textures, badges, LOD textures, etc.). Exported files are exported into a file with a **.hod** extension, stored in **Homeworld2/Data/Ship/<name of ship>**.

With regards to enabling a ship's LODs, LOD distance values must be added to the appropriate places in **Shiptuning.xls** (found in **Homeworld2/Data/Ship**) and an appropriate **.ship** file (placed in **Homeworld2/Data/Ship/<name of ship>)** must be generated.

We will go into further detail about these files and their directories as we get to them, but to summarize:

Homeworld2/Datasrc/Ship/<name of ship>

- ⇒ **<name of ship>.ma**
- ⇒ **associated textures**

Homeworld2/Data/Ship/

- ⇒ **Shiptuning.xls**

Homeworld2/Data/Ship/<name of ship>

- ⇒ **<name of ship>.hod**
- ⇒ **<name of ship>.ship**

10.2 Tutorial of the LOD and Weapons Effect Collision Mesh Process

Getting Started

Before we begin this tutorial, we have to make a couple of assumptions. Let's assume that you have made a hypothetical Hiigaran ship called 'Prototype' and that the base mesh is completed, and its textures are in place. We will also assume that this ship and is saved as **Hgn_Prototype.ma** in **Homeworld2/Datasrc/Ship/Hgn_Prototype** (which should also contain Hgn_Prototype's textures). Furthermore, let's also assume that it already has an entry in

Shiptuning.xls (in **Homeworld2/Data/Ship**) and that its exported .ship file is in **Homeworld2/Data/Ship/Hgn_Prototype**. Furthermore, we will assume that you know how to operate and use Maya to some degree as well as Photoshop.

The LOD Process

Typically, a ship that is at least Frigate sized should have 3 LODs (with one LOD sharing a goblin on/off state) to represent itself at different distances. Ships Corvette sized and under should have 4 LODs (with one LOD sharing a goblin on/off state). Unfortunately, due to different naming conventions that the artists and programmers used, there are some slight discrepancies as to how each of these LODs are referred to. For specifics about LOD polycounts, please refer to **Sections 2.1 or 10.7** in this document. This tutorial will assume the **programmers**' naming convention. Here is a table to outline these differences:

Artist	LOD1 w/ Goblins	LOD1 w/o Goblins	LOD2	LOD3	LOD4
Programmer	LOD0 w/ Goblins	LOD0 w/o Goblins	LOD1	LOD2	LOD3

Working with Layers

First of all, make sure that the drop box underneath 'File' in the Menu Bar says 'Modeling.' Once that is done, we can begin LODing.

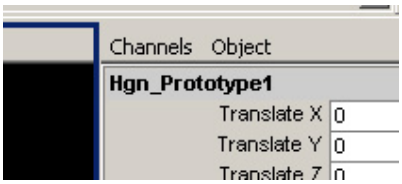
1. First of all, enable the Layer Bar in Maya, if you haven't done so already.
2. Beside the 'Display Layers' drop box, you will find a button with an icon that resembles three sheets of paper on top of each other.
3. Click on it, and you will note that a new layer called 'layer1' will appear. Double click on that and enter 'Ship' to replace 'layer1' as its name.
4. Now, select the main mesh of the ship as well as shift selecting any other parts that are not goblins, and then right click and hold on the layer that you just labeled 'Ship.'
5. Next, while still right clicking and holding on 'Ship', select 'Assign Selected' to assign the current selection of parts to the layer. You will notice that if you right click and hold on the 'Ship' layer and toggle visibility on and off, you will be able to toggle the visibility of the main mesh as well. Essentially, this will be your LOD0.

Unlike layers made for goblins (refer to the documentation about goblins), it doesn't *really* matter what you name these layers and what you assign to them as long as you know what you assigned to where. This makes it easier for you to differentiate between meshes, as well as reduce the amount of clutter you will inevitably get later on. Thus, we will repeat this process for every LOD mesh that is made if only for organization's sake.

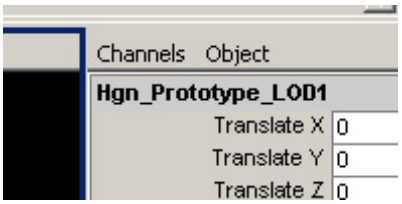
6. Next, using what you've learned so far, create a layer called 'LOD1'.
7. Shift select all parts of the ship that you want to have an LOD for (usually the main body and any other large outstanding features on the ship that would be highly visible) and duplicate these.
8. With these parts still selected, right click and hold on the 'LOD1' layer in the Layer Bar and select 'Assign Selected' to assign them to the layer.

9. Right click and hold on the 'Ship' layer, and toggle its visibility to off. Optionally, you can also toggle off the visibility for any other existing layer that you feel is cluttering up the display. In any case, the end result should be that only the things in layer 'LOD1' are visible.

Working with Meshes – LOD1



You will now have something to work with for the first LOD (LOD1). If you did the above steps correctly, everything that is in LOD1 should be a duplicate of something in layer 'Ship' (LOD0). Thus, assuming that the main mesh for our hypothetical ship was labeled as 'Hgn_Prototype' in LOD0, it will probably be named 'Hgn_Prototype1' in LOD1 as Maya automatically either assigns a number to a name of a duplicate part, or increases its number in the name by one (that is, if it was originally called 'Hgn_Prototype1', it will be duplicated as 'Hgn_Prototype2'). We don't want this.



Instead, we have to rename 'Hgn_Prototype1' to 'Hgn_Prototype_LOD1'. This tells the exporter to export that mesh as part of the LOD system. Any other parts in LOD1 also have to be named in this manner otherwise they will not show up as LOD1 parts and will instead be shown at LOD0 or another LOD altogether.

10. To go back to our example, make sure you have the Channel Box enabled and select the main mesh. Rename as described above.
11. Repeat this process for all parts that are in LOD1.

The next step in making the LOD1 mesh, is to start on the reduction of polys in the mesh. Assuming that you have a proficiency in Maya, this entails the merging of vertices, and deletion of edges. There are several things to keep in mind when making a LOD1 mesh (as well as any other level of LOD mesh):

- A balance must be made between visibility, and poly count. Although some 'popping' between LODs is to be expected, it should not become so apparent as to become distracting.
- Small edges, trenches, and holes can be safely filled or reduced, but try to avoid making large changes to large areas as much as possible. The key is to reduce and simplify so that the changes are not that easily seen from far away.
- You might also want to exaggerate certain features of the ship for ease of identification. This exaggeration should be subtle yet noticeable if looked for.
- Avoid texture warping as much as possible, and adjust UV coordinates as necessary if it occurs. If it takes a significant effort to fix these UV coordinates after the deletion of an edge or merging of vertices, don't delete those edges or merge those vertices.
- The poly count goal of LODs differs between ship classes. Generally speaking, the LODs for larger ships are allowed to have a higher poly count than LODs for smaller ships. Aim for as much reduction as possible. Some ships may not be as easily reduced due to their design, so keep reduction in mind when making the meshes in the first place.
- **If possible, avoid having to make and use textures for LOD2 and or 3**. Not only will this save you time in making a new texture, but also reduces the number of state

changes in the game, which will speed up the rendering time of the ship in the game. Unfortunately, the caveat is that sometimes this is admittedly impossible because by LOD2 and 3, the need for poly count reduction outweighs texture fidelity and ships. For example, small strike craft would probably require a new, separate LOD2 or 3 texture due to the fact that by the time they are at that LOD their poly counts must be so reduced that texture warping and general craziness is impossible to avoid. But by this time they are very tiny on screen so a low resolution texture is ok here, if needed.

- Remember to change the current perspective's Far Clip Plane (found in View -> Camera Attribute Editor… -> Far Clip Plane) so that you can see the mesh from farther out without having it getting clipped by the camera.

Working with Meshes – LOD2 and Beyond

For LODs 2 and beyond (the *Homeworld2* exporter supports up to 10), the same general steps apply for their creation:

- Create a layer for the LOD you want to do.
- Duplicate parts that you want to reduce the poly counts for from the previous LOD and then assign them into the newly created layer.
- Make sure that the name for each part has the appropriate suffix attached to the end (i.e. _LODX, where X represents the LOD number that you wish to use).

However, you may find that you need to create a new low-resolution texture to assign to these lower LODs due to texture warping as described in the last section.

Creating a Low Resolution Texture for LOD2 and Beyond

Essentially, what you have to do is build a new texture out of the existing original and assign new UV coordinates to this new texture so that the look approximates that of the original, but from a long distance away. Normally as sharp a texture as possible would be preferable, but because of the distances involved for the absolute lowest LOD, we can get away with using a smaller, lower resolution texture. The quickest way to do this is to take screen shots of the main ship with different layers of its texture activated and then combine them all into a new texture for use by LOD2 and beyond. This is probably the most difficult part of the LOD process to explain and do. However, it saves time in that you won't have to 'fix' UV coordinates because you will generate a new UV map in the process.

1. Save your work, and then save again under a different name (containing something like 'LODpose' in the name… for example 'Hgn_PrototypeLODPose') to differentiate this from the .ma you are going to export later on. This LODPose .ma file is used only for posing your ship model. Nothing else will be done with it besides using it to take screenshots.
2. Select and duplicate the main mesh and its other bit parts (except for goblins!) and then use the Move Tool to move it far away from the main mesh. This will serve as the top view from the Top perspective.
3. Select these duplicates, and duplicate them again. From the Menu Bar, select Edit -> Group and then use the Move Tool to drag it out and beside the original duplicate.
4. In the Channel Box, change the value of the 'RotateX' entry to 90. This will give you your 'front' view.
5. Duplicate the original duplicate, or duplicate the mesh that you just rotated, and then drag it out to any side and then fiddle with the 'Rotate…' values in the Channel Box to make

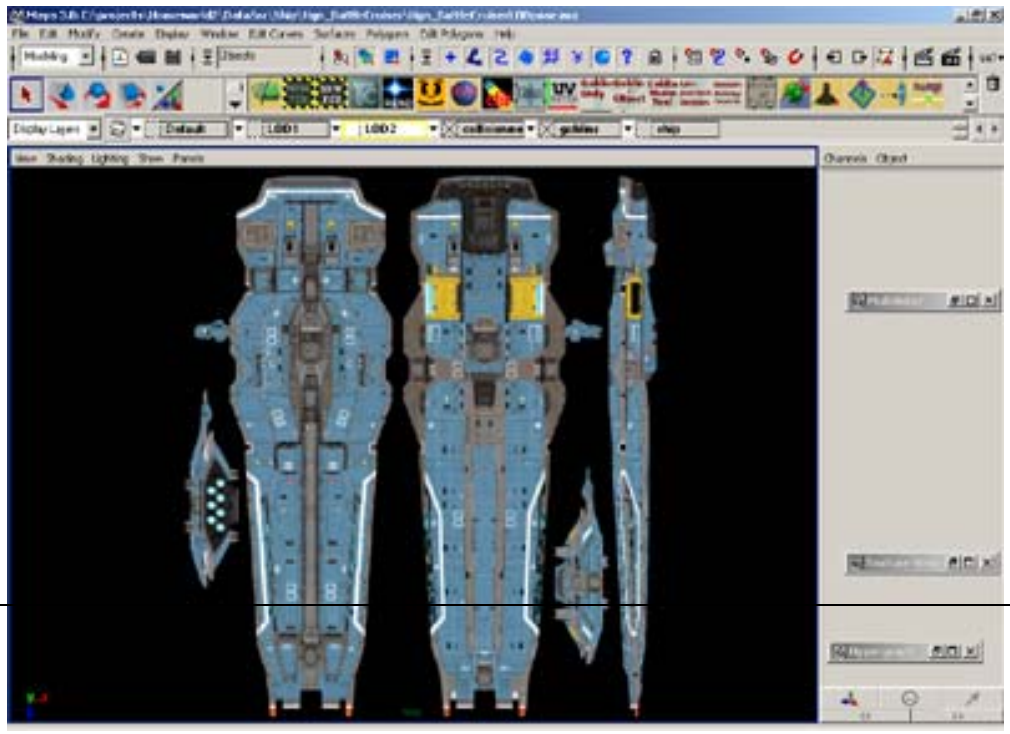
another view angle for your ship. Repeat this until the Top, Front, Side (left and right if necessary), Back, and Bottom views are accounted for.

- **Note for ships that have internal hangar bays, etc.:**
    - Try to select the faces of the walls in the bay and then, in the Menu Bar, go to Edit Polygons -> Extract. Then go to Edit Polygons -> Separate. This will separate the faces from the mesh, but without moving them out of place. Remember, you are not exporting this .ma file so it's ok to do this. Duplicate these separated faces and then drag them out to the side so that they are visible. Rotate accordingly so that they can be seen from Maya's Top camera perspective.

6. Switch the camera perspective that you are using to Top and arrange the different poses so that they roughly cover a square-ish area. Center the camera over these objects. Turn off any grids that are enabled.
7. Go to 'Hypershade…', and open up the attribute editor for each used texture. Click on the button beside the 'Color' slider under 'Common Material Attributes' and then change the filename of the texture from the export version to that of the original, unexported version. Once this is done, change its incandescence to maximum. This will make it so that your ship doesn't have any shadows cast on it so you can take a better screenshot that has its original colors.

IMPORTANT NOTE: Make sure that the textures that you are working do not contain things that are off the canvas. Using such a texture crashes Maya. To rectify this in Photoshop, click on Select -> All to select the entire texture. Then click on Image -> Crop. Finally, textures have to have dimensions that are factors of 2 in order to work.

8. Make sure your camera focus distance includes all of the visible portions of your mesh when viewed from the top. Your screen should then look something like this but with your ship instead (a Hgn_BattleCruiser is displayed here as an example):

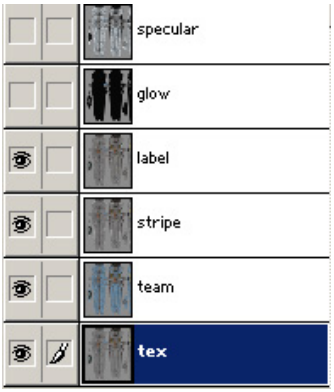


9. Next, go into Photoshop and toggle off the visibility for all the layers except for the tex layer for all necessary textures (i.e. side textures, front textures, etc.).

10. Save the changes and then go into Maya and click on the 'Refresh Textures' button (labeled 'New Tex') in the Relic Tools. Your ship should now be displaying the only visible layer (in this case, the tex layer).

11. Go back to Maya and with the ships centered however you want on screen, do an ALT-PrintScreen to take a screenshot of what you have. Go back into Photoshop and create a new image and paste the screenshot into it. It should make a new layer in this new image.
12. Rename this layer in your new image to 'tex.'
13. Next, go back to the ship's original textures and then turn off the visibility of the 'tex' layers for all the textures. Turn on the visibility of the layer above it. This will usually be the 'team' layer.
14. Repeat steps 10 and 11. This time you should have a new layer with the 'team' layer on the ship. Keep in mind that sometimes you will have to keep the 'texture layer' visible in order for the 'team' layer to show up properly. Also make sure that there are no background or reference layers present in the original layer, as they will show up on the ship's mesh underneath the 'team' layer.
15. Rename this layer in your new image to 'team.'
16. Go to Photoshop and turn off the visibility for the 'team' layer for all of the ship's original textures and turn on the visibility for the 'stripe' layer for them too.
17. Repeat steps 10 and 11. You will now have a new layer in your new texture that will be your 'stripe' layer.
18. Rename this layer in your new image to 'stripe.'
19. Turn off the visibility for the 'stripe' layer and turn on the visibility for the 'label' layer for all of the ship's original textures.
20. Repeat steps 10 and 11. You will now have a new layer in your new texture that will be your 'label' layer.
21. Rename this layer in your new image to 'label.'
22. Turn off the visibility for the 'label' layer and turn on the visibility for the 'glow' layer for all of the ship's original textures.
23. At this point, you may want to change the background color of Maya to be pitch black but it's entirely optional. Anyway, repeat steps 10 and 11.
24. Rename this layer in your new image to 'glow.'

25. Turn off the visibility for the glow layer and turn on the visibility for the 'specular' layer for all of the ship's original textures. Repeat steps 10 and 11.
26. Rename this layer in your new image to 'specular'.

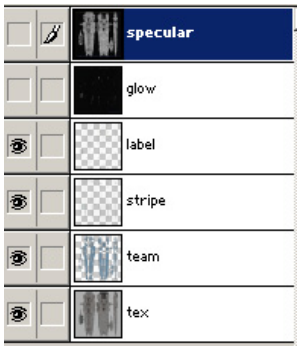


Finally, you now have a rough texture to work with to make a LOD2/3 texture. Save your work!! Your texture will have the following layers in this order, starting from bottom up: tex, team, stripe, label, glow, and specular. It should look something like the picture to the left.

27. Use the Crop Tool in the Tools to crop the image such that the only things visible are the ship poses. Make sure this is roughly a square. To fine tune this canvas into a perfect square shape, go to Image -> Canvas Size and adjust accordingly.

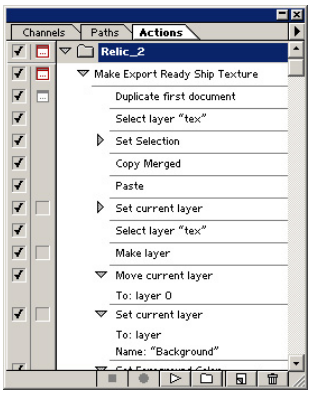
28. Next, with the visibility turned off for all layers except the 'tex' layer, Eyedrop the color of the background in the 'tex' layer and then, while using that same color, fill in any gaps in that same layer created by the canvas resizing. Remove any unwanted imagery and fill it with the background color.
29. Now that you've done the 'tex' layer, it's time to do the 'team' layer. Turn off the visibility for 'tex' and turn on the visibility for the 'team' layer. Use Select -> Color Range… and then select the team color for the ship. Adjust the 'fuzziness' level to adjust the amount of that color you want to be picked. In our case, try to get as much of the color as possible. Click on OK and then invert the selection. Delete. Now, only the team color should be remaining in this layer. Another alternative is to use Color Range to pick out the colors that *aren't* the team color for your ship and delete them, leaving the team color untouched.
30. In the Layers there should be a drop box that determines the properties of any selected layer. With 'team' still selected, click on the drop box button and select 'Overlay' from the menu.
31. Turn off the visibility for the 'team' layer and turn on the visibility for the 'stripe' layer. Repeat steps 30 and 31, but for the 'stripe' layer.
32. Turn off the visibility for the 'stripe' layer and turn on the visibility for the 'label' layer. Repeat step 30, but for the 'label' layer. In this case, try to delete everything that is not a label. Also, keep this layer's property as 'Normal'
33. Turn off the visibility for the 'label' layer and turn on the visibility for the 'glow' layer. For the 'glow' layer, make sure everything that is shown is in black or white. You may need to desaturate the colors (Image -> Adjust -> Desaturate) in this layer.

34. Fill in the background color for the 'glow' layer with black.
35. Turn off the visibility for the 'glow' layer and turn on the visibility for the 'specular' layer. Repeat steps 34 and 35 for the 'specular' layer.



Now that you're finished adjusting the layers, we will now set the final visibility settings for the layers. The final visibilities for '**tex,**' '**team**,' '**stripe**,' and '**label**' layers should be **on**. The final visibilities for the '**glow**' and '**specular**' layer should be **off**. Refer to the picture on the left. Now your image is more of a texture. Save your texture under another filename if you wish. Your LOD2/3 texture is now ready for resizing and export! More on ship texture layers can be found in Section 8.6.

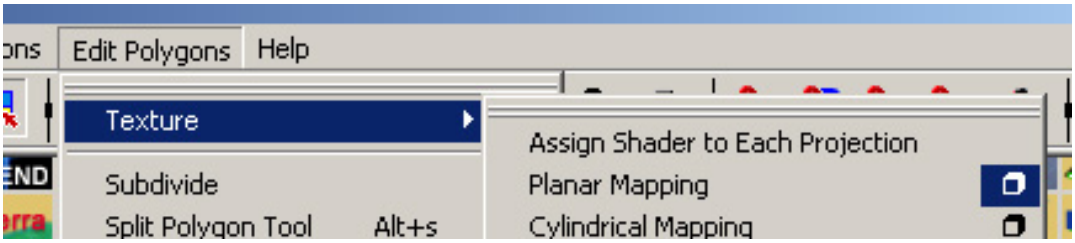
36. To resize the texture, go to Image -> Image Size and then change the pixel dimensions. Remember to use numbers that are factors of 2. You might want to consider using different dimensions, depending on the size of the ship that this texture is for, and how visible it is in the game. Larger ships can and should use larger textures, like 128x128 since they stay large longer on the screen. Smaller ships like strike craft can get away with using 64x64 textures. For frigates, use your judgment to decide between 128x128 or 64x64. Save this texture as '<name of ship>_LOD2_TEX.psd' or some other similar variant. In our hypothetical example, it would be called 'Hgn_Prototype_LOD2_TEX.psd'.
37. Now to export your texture… Make sure that you have the Photoshop Export Macro installed. Click on the 'Actions' tab and then under 'Relic_2,' select 'Make Export Ready Ship Texture.' Click on the 'Play' button (the 'Play Current Selection' button in the 'Actions' tab) and enter '<name of ship>_LOD2_EXPORT' in the box. Click on 'Save As…' and save as a '<name of ship>_LOD2_EXPORT.psd'.



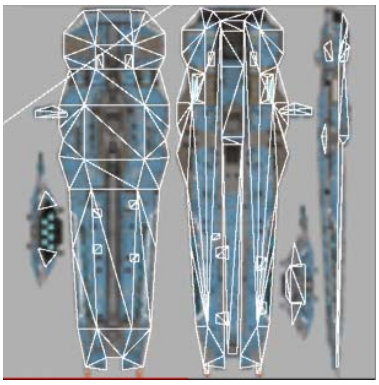
Your texture is now done. However, to use it we must apply it to any LOD2/3 meshes that you've made. Basically what we do is project the texture onto where we want on the meshes.

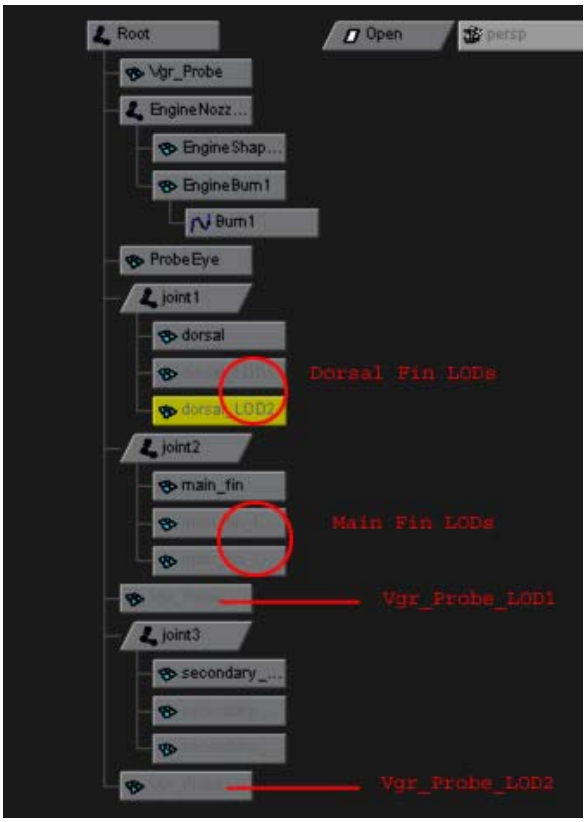
38. Go back to Maya, and load up the latest .ma that wasn't used for LOD posing. Make sure that the visibilities for the Maya layers is set so that only a single LOD2/3 mesh is visible.
39. Create a new HW2 material by clicking on the 'Create HW2 Material' button in the Relic Tool Bar. In Hypershade, open up the Attribute Editor for this new material and assign the export texture to it (by clicking on the box beside the 'Color' slider and browsing). Alternatively, you can duplicate an existing material and replace its current texture with the export texture. Either way, rename the material to make it easier to tell apart the LOD2/3 texture from the other ones.
40. Highlight the LOD2/3 parts that you wish to assign this texture to, and then in Hypershade, right click and hold on the LOD2/3 export material and select 'Assign Material to Selection.' This will assign the new material to the LOD2/3 mesh.

41. Activate the 'Select By Component: Faces' option (F11). This will allow you to choose the faces of the mesh that you wish to project onto.
42. Face select a whole side of the LOD2/3 mesh according to what you have in the export texture ('<name of ship>_LOD2_EXPORT'). For example, try selecting all the faces of your mesh that would correspond to the 'top' view represented in the export texture.
43. Click on Edit Polygons -> Texture and then select the box symbol beside the 'Planar Mapping' option (see picture below). A window titled 'Polygon Planar Projection Options.'



44. Because this example is dealing with the 'top' view of the mesh, and the Y-axis in Maya corresponds to either looking straight down or straight up, make sure the Y-axis radio 'Mapping Direction' button is highlighted in 'Polygon Planar Projection Options.' Click on 'Apply.'
45. You'll notice that the selected faces are now highlighted as one group. Bring up Window -> Texture View… and you'll notice that they are highlighted in this window too. Drag these off to anywhere that is unoccupied. We will come back later to place these UV coordinates in their correct places.
46. Repeat steps 43-46 as necessary, making sure that the faces that you highlight receive the correct 'Mapping Direction' (i.e. side view usually gets the X-axis direction, front/back gets the Z-axis direction).
47. When you are finished projecting all the sides, go into the Texture View window and right click and hold on the display and then select 'UV' from the menu that pops up. Drag select each group of UVs and then using things like the Move Tool, Rotate UVs, and Move Component (to scale) to move, rotate, and scale these UVs to the desired location and position. Make sure that your UVs correspond closely to the way the ship looks in the previous LODs. Eventually, you will have something that looks like what is on the right. Compare it with the model in 3D view. Make adjustments where needed.





48. Finally, go into Hypergraph and join all the relevant LODs to the root node. For things like arms and stuff that are linked to other joints/nodes that are linked to the root node, link the LODs to their respective joints instead (otherwise weird animation problems may occur). For your reference, we will display the Hypergraph for the Vgr_Probe to illustrate where LODs go in relation to the root. Also, make sure that all relevant textures are in the appropriate directory (i.e. Homeworld2/Datasrc/Ship/<name of ship>).

Congratulations!! You have now made LODs for your ship, including its lowest LOD that contains a lower resolution texture. You can now continue to Section 11.4 (Exporting) to learn how to export your ship, or continue onto the next section to learn more about collision meshes and how to create them.

10.3 Making Weapons Effect Collision Meshes

Weapons effect collision meshes are used to determine where a weapon hits and where the resulting hit effect gets spawned. Ideally, such a mesh would stick as closely as possible to the shape of the ship underneath. This is to ensure that weapons hit and their effects spawn as closely to the hull as possible.

If there is no such mesh on a ship, the game will automatically generate a weapons effect collision box from the bounding volume of the ship itself (taken from its maximum length, width, and height). This, of course, is not desirable since most ships are not square or rectangular in shape and resulting hits and spawned effects would be generated in the empty volumes around the ship.

The creation of a collision mesh is actually quite easy, as it is similar to making an LOD mesh. Although they won't be visible, try to make these meshes as low poly as possible. Also, there are two ways you can make such a mesh. You can make a duplicate of the original ship mesh, assign a Lambert texture to it, and then reduce until you get a low poly mesh that approximates the shape of the original. The other way is to use an existing lower poly LOD mesh and assign a Lambert texture to it. This second method can save you some time, but be prepared to make some adjustments to the mesh in order to accommodate the original mesh's shape (since the LOD meshes are themselves approximations of the original too).

Once this mesh is made, rename the mesh to CM_<name of ship>. In our hypothetical Hgn_Prototype ship's case, we would call its mesh 'CM_Hgn_Prototype.' This is linked directly to the root node of the mesh. Any other parts that need collision meshes for the same ship must have a CM_ prefix and must be linked to the same joint/node of the part the mesh is for. Note that this arrangement is similar to how LODs are hooked up to the root node (**see step 48 in Section 2.2: The LOD Process**).

10.4 Exporting

Dealing with .hods

Now that you have a ship's LODs (and possibly its weapons effect collision mesh as well), we will have to export the ship into a .hod file for it to work in game.

1. Make sure you select Edit -> Delete All by Type -> History to delete unnecessary history that may clutter up the file. Save your work.
2. Click on the box beside File -> Export All. This will open the 'Export All Options' window.
3. Beside where it says 'File Type,' there should be a drop box. Click on it, and select '**hod**' from the list.
4. Make sure that the '**Ship**' radio button is selected in the 'Export Type' section.
5. Make sure that 'DXT5(rgba)' is selected in 'Texture Options.'
6. Make sure that in the 'Optimization Options' that Triangle List is selected and Merge is selected.
7. Once this is done, click on the 'Export All' button.
8. Select the directory the .hod is going into. In the case of our hypothetical ship, we will place it in Homeworld2/Data/Ship/<name of ship>. Thus, we will export our Hgn_Prototype.hod file into Homeworld2/Data/Ship/Hgn_Prototype.

10.5 Adding LOD Values to Shiptuning.xls

The Shiptuning.xls file can be found in the directory 'Homeworld2/Data/Ship'. You will require Excel with enabled macros to view it. When you do load up Shiptuning.xls, you will find columns that display the various stats for all of the ships in the game, playable and non-playable. Let's pretend that Hgn_Prototype has similar values to Hgn_IonCannonFrigate.

1. Highlight Hiigaran Ion Cannon Frigate's column, and then duplicate it. Rename the duplicate column to 'Hiigaran Prototype'.
2. Scroll down through the stats until you get to the section labeled 'UI/Rendering' and then note where the LOD distance values and goblin fade and disappear distance values are entered.
3. Enter desired distance values. Lower poly LODs should activate when relatively far away from a ship. To use the Hiigaran Ion Cannon Frigate column as an example, the LOD1 (**Note: Shiptuning.xls lists the LODs as per artist definition. See Section 11.2 for more information**) for the Hgn_IonCannonFrigate will kick in at 800 meters. The LOD2 for the Hgn_IonCannonFrigate will kick in at 1300 meters. Since there is no LOD3 mesh for the Ion Cannon Frigate, it was left blank. At 815 meters, the ship will switch from LOD1 to LOD0, and at 1315 meters, the ship will switch from LOD2 to LOD1. Adjust the values for Hiigaran Prototype as you see fit.

4. When you are done, hit the 'Create/Update/Export Ship Data' button (make sure the .ship file is not read-only).
5. A .ship file called Hgn_Prototype.ship will be generated and automatically placed into Homeworld2/Data/Ship/Hgn_Prototype (which is also automatically generated).
6. Save your work.

And that is how you add and enable a ship's LODs and its weapon effects collision mesh for use in the game!

10.6 Summary

Making LODs and weapons effect collision meshes can be summarized as follows:

1. Get the main mesh, make sure all the associated textures are in the same directory (ie. Homeworld2/Datasrc/Ship/<name of ship>).
2. Make a layer to store the main mesh in, and duplicate parts as needed. Make a new layer, call it LOD1 and assign these duplicates to it.
3. Perform poly reduction magic on this mesh. Add a _LOD1 suffix to all relevant parts.
4. Repeat 2 and 3 for LOD2, but duplicate the LOD2 mesh instead. Repeat in the same manner for any subsequent LODs (ie. for strike craft).
5. If needed, make a new low-resolution texture for LOD2
    a. Using visibility toggles, make an all new texture by taking screenshots of the ship displaying various layers. Use these shots to create layers for the new texture.
    b. Make sure that this new texture conforms to the texture format and then export.
6. Create a weapons effect collision mesh from scratch (ie. start from the original mesh) or tweak an existing LOD while remembering to assign it a Lambert shader and a CM_ prefix.
7. Export into a .hod file of the ship into Homeworld2/Data/Ship/<name of ship>.
8. Add LOD values to Shiptuning.xls found in Homeworld2/Data/Ship.
9. Export the .ship file for your ship into Homeworld2/Data/Ship/<name of ship>.

10.7 General LOD Poly Count Guidelines

This is the ideal poly count for objects in the game. While some of these numbers may not be realistically possible, effort must be made to reduce the count as much as possible. These figures were generated in September 2002.

Base Polygon Allocation – September 2002

Class	%Base Budget	Poly Budget	Base Mesh	Goblins	% Goblins
Fighter	25%	825	660	165	20%
Corvette	50%	1650	825	825	50%
Frigate	100%	3300	1320	1980	60%
Cruiser/Carrier	120%	3960	1346	2614	66%
Mothership	175%	5775	1964	3812	66%

LOD Poly Count

Class	LOD0	LOD0 w/Goblins	LOD1	LOD2	LOD3
Fighter	825	660	220	100	~25
Corvette	1650	825	270	100	~25
Frigate	3300	1320	450	150	N/A
Cruiser/Carrier	3960	1346	470	200	N/A
Mothership	5775	1964	650	500	N/A

11 Docking Paths

11.1 Maya

Many things are done in Maya with the help of various custom Maya plugins. These are designed to enable a developer to add additional content to the base *Homeworld2* ship meshes. In the case of the ship launch/dock/latch path pipeline, plugins are used to add and tweak these. Specifics on what each function pertaining to these paths are given in the HW2_DockLaunchLatchPathsRDNHelp.doc. **Also note that sometimes paths fail to show up upon initial loading of a ship**. **This is a Maya bug and the quickest way to see these paths again is simply to just do a quick save over the original**. Once this is done, the paths should become visible again.

11.2 File Extensions and Directories (Folders)

There are a few general things that you must keep in mind regarding file extensions and directories before you begin the process of adding launch/dock/latch paths to a ship (or any other customization process).

Ships in *Homeworld2* begin as meshes saved as Maya ASCII files (**.ma** extension). These .ma files are stored in **Homeworld2/Datasrc/Ship/<name of ship>**, along with any associated textures (such as hull textures, badges, LOD textures, etc.). Exported files are exported into a file with a **.hod** extension, stored in **Homeworld2/Data/Ship/<name of ship>**.

With regards to enabling a ship's launch/dock/latch paths, as mentioned earlier, certain values must be set in the appropriate places in **Shiptuning.xls** (found in **Homeworld2/Data/Ship**) and an appropriate **.ship** file (placed in **Homeworld2/Data/Ship/<name of ship>)** must be generated.

To summarize the files (and their directories) affected by launch/dock/latch paths:

Homeworld2/Datasrc/Ship/<name of ship>

- ⇒ **<name of ship>.ma**
- ⇒ **associated textures**

Homeworld2/Data/Ship/

- ⇒ **Shiptuning.xls**

Homeworld2/Data/Ship/<name of ship>

- ⇒ **<name of ship>.hod**
- ⇒ **<name of ship>.ship**

11.3 Explanation of the Differences Between Types of Paths

Docking paths are paths that ships follow to 'dock' **into a hangar bay (or something similar)** of another ship. Usually this means a smaller craft docking into a larger ship. For example: a fighter going inside a carrier to repair or retire.

Launch paths are paths that ships follow to 'launch' from another ship. This usually means a smaller craft launching **from a hangar bay (or something similar)** of another ship. For example: a freshly built frigate launching from a mothership. These are differentiated from docking paths through the use of an 'exit path' flag while making the path in Maya. **Note that if no launch path exists for a specific class of ship, and if the player attempts to launch that specific type of ship from a parent ship, the specific ship will appear via hyperspace**.

Launch path and docking path use is determined by the use of flags that indicate what classes of ships can use them. **Enabling a ship to produce other ships of certain kinds though, is another topic altogether and is covered in the HW2_BuildAndResearchScripting.doc**. However, note that a ship doesn't have to be a production ship in order to have other ships launching from it or docking into it. The best example of this are the races' battlecruisers, which can launch and dock a limited number of strike craft.

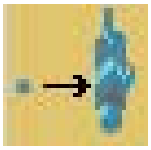
Latch paths are usually used by resource collectors to 'latch' on to **and** off of a ship for dropping off their resources. For example: a Hiigaran resource collector follows a latch path to latch on to a mobile refinery to drop off resources. A separate latch path with an 'exit path' flag is then used for the latch path that the resource collector follows to unlatch from the mobile refinery. Another example of latch path use would be for ultracaps docking on to certain ships in the *Homeworld2* single player campaign. Essentially, latch paths are used by ships that need to dock with something, but don't disappear into a hangar bay or something similar.

11.4 Tutorial on Paths

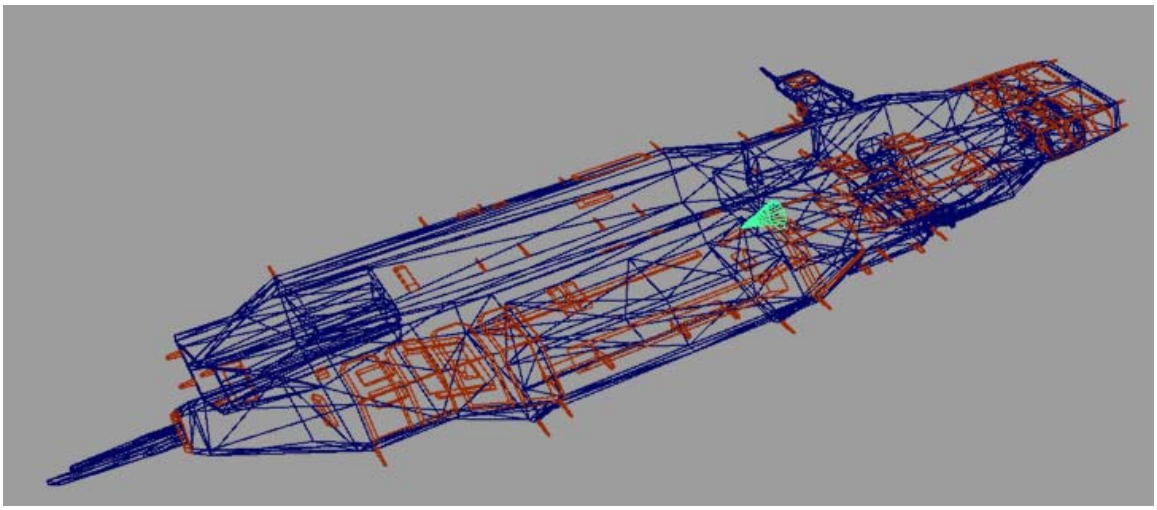
Using a base Hiigaran Carrier without paths as our demonstration ship, we will now show you how to make your own launch/dock/latch paths for your ships. This tutorial assumes that you have a working knowledge of Maya and that all plugins are installed. Paths are made up of points that are defined through **keyframes** and the flags that are defined within these keyframes.

How to Make Launch Paths – A Fighter Launch Path Example

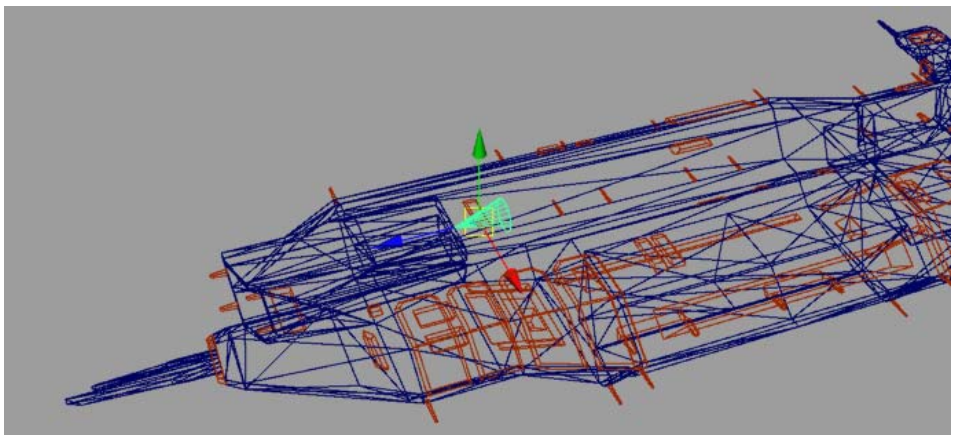
1. Fire up Maya and load up the ship in question.
2. Make sure you have your ship in perspective view and set the view's shading to 'wireframe'. To do this, click on Shading -> Wireframe in the viewport's Menu Bar. This will make your ship transparent so you can see where your paths are going.
3. Make sure that you have the Time and Range Sliders showing in Maya. To do this, go to Display -> UI Elements and check off the appropriate options.
4. Make sure that the working units for time is set to NTSC(30fps). To do this, go to Window -> Settings/Preferences -> Preferences… and in the Preferences window, select 'Settings' from the column on the left. In the 'Working Units' section, click on the drop box associated with 'Time' and select NTSC(30fps).
5. Slide the Time Slider to 1 (not 0!).

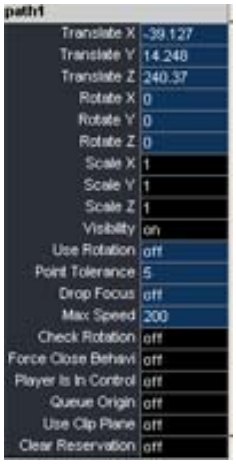


6. Click on the '**Docking Path**' icon (shown on the left) in the Relic Tools Shelf. This will create a docking path cone shape in the middle of the carrier. You can rescale this if you like, but because the game only looks at the points along a path to determine how the ship moves along, it doesn't really matter. It's position in relation to the carrier should be like this:



7. Making sure that the cone stays selected, look at the Channel Box on the right hand side and type in 'on' beside 'Exit' and 'Fighters'. Alternatively, bring up its Attribute Editor and then open up the Extra Attributes section. Check off the check boxes labeled 'Exit' and 'Fighters'.
8. In the Channel Box, enter 'on' for Use Rotation.
9. Set Point Tolerance to '5' or some other low number, but make sure the value is not '0'.
10. Set the Max Speed to '200'.
11. Move the cone to the desired start position. Let's make this path originate from just inside the top hangar bay of the carrier:





12. Select and drag the entire selection of attributes and values in the Channel Box that are under ‘path1’ (don’t select the things that are under ‘SHAPES’). It should look similar to the picture on the right.
13. Right click and hold on the selection. A menu should come up. Select ‘Key Selected’. This will make the first position of this launch path for fighters originate from just inside the top hangar bay with a speed of 200 at keyframe 1. You will notice that the position of this first keyframe is indicated on the Time Slider by a small, thin red line.
14. Let’s move the cone out so that any ships following this path will exit the carrier. Slide the Time Slider to 10. Move the path cone a fair distance away from the carrier, along the axis parallel to the facing of the hangar bay (in this case, along the Z-axis). **This point must be far away from the carrier otherwise the ships following the path will be stuck due to interference from the avoidance box around the carrier**.
15. Change Clear Reservation to ‘on’. This means that once a ship passes this point, ships that are launching along the same path can start their launch from the beginning point. Also, change the Point Tolerance to something higher, like ‘50’ or ‘100’.
16. Repeat step 12. Select ‘Key Selected’. If you move your slider out of the way, you will notice that a second thin, red line has appeared on the Time Slider at 10.
17. Try sliding the time slider back and forth to see the launch path of the fighters. You will notice that it moves out from the carrier along the Z-axis, as expected:

And that’s how you do a simple launch path! This specific path was easy because it was in a straight line. However, more complex paths can easily be made. At the desired keyframe (or point along the path, if you wish to call it that instead), all you have to do is make sure the cone is in the desired position and with the correct attributes entered and collected, right click on the selected attributes and then select ‘Key Selected’. Also, you might want to increase the point tolerance.

How to Make Docking Paths – A Fighter Docking Path Example

The process used to make a docking path is slightly more complex than making a launch path. You have to specify a queue point in the beginning of the path in order to tell ships where to queue up for that path, as well as make sure that docking ships don't act weirdly when trying to dock. Also, docking is very much affected by collision avoidance. The first point that the ship actually uses to go along the path is the one that is put in after the queue point. Let's make a docking path for fighters as an example.

1. To start, deselect anything that you may have selected.
2. Slide the Time Slider back to 1.
3. Hit the **Docking Path** icon in the Relic Tools Shelf.
4. Move the path cone to where you want the ships to start lining up. Make sure that it is sufficiently far enough from the target ship. It is advisable to put this path cone to the rear of the ship, rather than coming in at the side. This is because some ships are poor at moving side to side (lateral movement) and may not be able to move fast enough to make it to points along its dock path while the target ship is moving.
5. With the cone still selected, go to the Channel Box and type in 'on' in the box beside 'Fighters'. Alternatively you can bring up the Attribute Editor, and expand the Extra Attributes section and check off the check box beside 'Fighters'.
6. In the Channel Box, enter 'on' for Use Rotation.
7. Add '100' for Point Tolerance.
8. Enter 'on' where it says Player Is In Control.
9. Enter 'on' where it says Queue Origin.
10. Drag select the entire contents of the top part of the Channel Box (see step 12 in **Section 2.1**).
11. Right click and hold the selection. Select 'Key Selected' from the menu that comes up. This will create a queue point for fighters at keyframe 1.
12. Move the Time Slider to 10.
13. Move the path cone to where you want the ship to start its docking run. In our case, lets move the path cone a bit forward.
14. Change the Queue Origin to 'off'.
15. Change the Max Speed to 200.
16. Drag select the entire contents of the top part of the Channel Box again.
17. Right click and hold the selection. Select 'Key Selected' from the menu that comes up. This will create the first point in the docking path for fighters at keyframe 10.
18. Move the Time Slider to 20 or any other number higher than 10.
19. Move the path cone to the next desired position.
20. Change the Point Tolerance or Max Speed to something lower. Turn off Player Is In Control if desired.
21. Repeat steps 16 and 17. This will create the next point at wherever you moved the Time Slider in step 18.

22. Repeat steps 18 - 21 as desired.
23. Make sure that at the last position, you have Clear Reservation set to 'on' before you drag select and set a keyframe.

And that's how you add a docking path for fighters. For ships with a larger mass (like frigates or supercaps), you can make ships move sideways by using 'Force Close Behaviour' in conjunction with sideways movement. 'Check Rotation' is also useful when you want to make sure a ship is pointing in the correct direction before continuing. Furthermore, and especially in the case of larger ships, you might want to consider enabling 'Use Clip Plane' at the point where a ship passes through the wall if that ship is larger than the ship it is docking with (otherwise it will poke through the other side). For more information about these commands, refer to Section 11.8).

How to Make Latch Paths – Sample Resource Collector Latch Path

Making latch paths is different in that we usually start by making the incoming path first. That being said, creating an incoming path is exactly the same as creating a docking path, except that the last point in the path ends up at an external location on the outside of a target ship rather than inside a hangar bay. **An outgoing path must be made in order for a latching ship to unlatch from a target ship and move away**. This path is made the same way as a launch path, except that **it must start at exactly the same location as the last point of the incoming path** (but at frame 1). For both of these situations, you will need to check off the 'Latch Path' box in the Extra Attributes for the paths in question. Let's make a sample resource collector latch path for a Vgr_ResourceController. We'll do one side (these instructions would apply to either side because we'll make the paths symmetrical).

Making a Latch Entry Path

1. Let's start off with the queue point. We'll assume that nothing is selected and no other paths have been made. **Make sure the Time Slider is set to keyframe 1**.
2. Click on the 'Docking Path' icon (see Section 2.1 Step 6).
3. Move the path cone to where you want the ships to start lining up. Make sure that it is sufficiently far enough from the Vgr_ResourceController so that it doesn't conflict with the Vgr_ResourceController's avoidance box. It's advisable to move this origin point to somewhere behind the ship. This is because collectors are poor at moving side to side (lateral movement) and may not be able to move fast enough to make it to points along its dock path while the target ship is moving.
4. With the cone still selected, go to the Channel Box and enter 'on' in the box beside 'Utility'. **Enter 'on' for Latch Path as well**. Alternatively you can bring up the Attribute Editor, and expand the Extra Attributes section and check off the check boxes beside 'Utility' and 'Latch Path'.
5. In the Channel Box, enter 'on' for Use Rotation.
6. Enter '100' for Point Tolerance, and something appropriate in Max Speed for collectors.
7. Enter 'on' where it says Player Is In Control.
8. Enter 'on' where it says Queue Origin.
9. Drag select the entire contents of the top part of the Channel Box (see step 12 in **Section 2.1**).
10. Right click and hold the selection. Select 'Key Selected' from the menu that comes up. This will create a queue point for fighters at keyframe 1.

11. Move the Time Slider to 10.
12. Move the path cone to where you want the resource collector to start its RU drop-off run. In our case, lets move the path cone a bit forward. However, make sure it is still far away from the target ship, to avoid interfering with the avoidance box of the controller.
13. Change the Queue Origin to 'off'.
14. Change the Max Speed to something appropriate for a collector.
15. Drag select the entire contents of the top part of the Channel Box again.
16. Right click and hold the selection. Select 'Key Selected' from the menu that comes up. This will create the first point in the latch path for collectors at keyframe 10.
17. Move the Time Slider to 20 or any other number higher than 10.
18. Move the path cone to the next desired position.
19. Change the Point Tolerance or Max Speed to something lower. Turn off Player Is In Control if desired.
20. Repeat steps 15 and 16. This will create the next point at wherever you moved the Time Slider in step 17.
21. Repeat steps 17 - 20 as desired. In our case, repeat until the cone gets inside the docking clamps of the Vgr_ResourceCollector. **Make a note of its XYZ and rotation values at this point**.

Making a Latch Exit Path

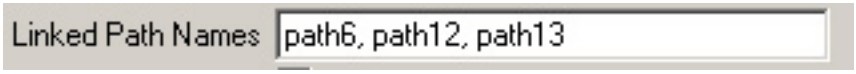
22. Deselect everything.
23. Slide the Time Slider back to 1. Click on the 'Docking Path' icon again. This will make a new cone.
24. **Manually enter the XYZ and rotation values that you just copied down**.
25. With the cone still selected, go to the Channel Box and enter 'on' in the box beside 'Utility'. Enter 'on' in Latch Path and enter 'on' in Exit as well. Alternatively you can bring up the Attribute Editor, and expand the Extra Attributes section and check off the check boxes beside 'Utility', 'Latch Path', and 'Exit'.
26. In the Channel Box, enter 'on' for Use Rotation.
27. Set Point Tolerance to '5' or some other low number, but make sure the value is not '0'. You might also want to set Force Close Behavior to 'on'.
28. Give Max Speed an initial value that would be logical for a ship of a collector's mass.
29. Drag select the entire contents of the top part of the Channel Box (see step 12 in **Section 2.1**).
30. Right click and hold the selection. Select 'Key Selected' from the menu that comes up. This will create the first point in the exit latch path for collectors at keyframe 1.
31. Move the Time Slider to 10 or some higher number.
32. Move the path cone to the next desired position. For our purposes, let's move it straight to the side along the X-axis, away from the controller. Make sure that subsequent points are well clear of the Vgr_Controller's avoidance box.

33. Change the Point Tolerance or Max Speed to something higher. Somewhere along the line during this path's creation you may want to enable Player Is In Control. Use your judgment.
34. Drag select the entire contents of the top part of the Channel Box (see step 12 in **Section 2.1**).
35. Right click and hold the selection. Select 'Key Selected' from the menu that comes up. This will create the first point in the exit latch path for collectors at keyframe 10 (or another number that the Time Slider is at).
36. Repeat 31 – 35 as desired. Remember to move the Time Slider before moving a cone and setting its keyframe.

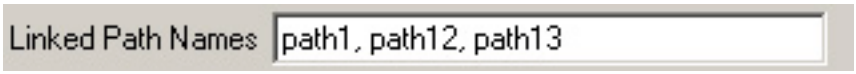
And that's how you create a path for resource collectors!

11.5 Path Sharing

Sometimes there are several launch/docking paths sharing the same bay. Normally you don't want them to collide and conflict with each other if they are too close together (i.e. a frigate that has just been built is coming out of a bay but because it takes up most of the bay, another frigate that has just been retired will collide or interfere with the first frigate as it comes out). The solution to this is to enter the names of the paths that it shares space with in 'Linked Path Names' in the path cone's Extra Attributes in its Attribute Editor. Here is an example:



Let's pretend that the path that this is for belongs to 'path1'. The picture above indicates that this 'path1' is linked to paths 6, 12, and 13. If you go to path6 and examine its Linked Path Names, you'll notice:



Note that path1 is listed, but not itself. Likewise, path12 and path13 will be similar in that all the other paths will be listed except for the path in question. Used in conjunction with 'Clear Reservation' you can solve a lot of docking and launching path problems. Latch paths don't normally need path sharing since they are usually far apart enough to not interfere with each other.

11.6 Exporting It All

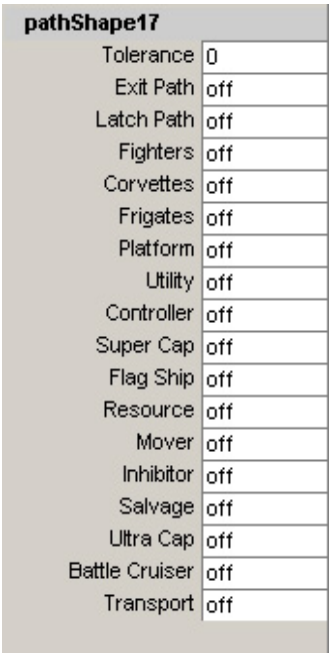
1. Make sure you select Edit -> Delete All by Type -> History to delete unnecessary history that may clutter up the file. Save your work.
2. Click on the box beside File -> Export All. This will open the 'Export All Options' window.
3. Beside where it says 'File Type,' there should be a drop box. Click on it, and select '**hod**' from the list.
4. Make sure that the '**Ship**' radio button is selected in the 'Export Type' section.
5. Make sure that 'DXT5(rgba)' is selected in 'Texture Options.'
6. Make sure that in the 'Optimization Options' that Triangle List is selected and Merge is selected.
7. Once this is done, click on the 'Export All' button.
8. Select the directory the .hod is going into. In the case of our test Hiigaran Carrier, we will place its Hgn_Carrier.hod in 'Homeworld2/Data/Ship/Hgn_Carrier'. In the case of the Vaygr Resource Controller, we will place its Vgr_ResourceController.hod in 'Homeworld2/Data/Ship/Vgr_ResourceController'.

11.7 Summary - Tips

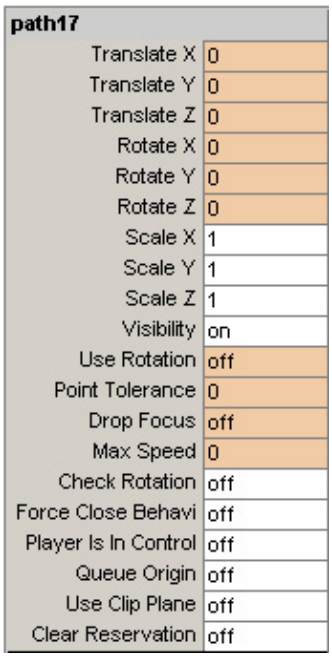
1. Docking paths are for ships to follow to dock into a hangar bay of a larger ship. Launch paths are the opposite – these allow ships to launch from a hangar bay of a larger ship.
2. Latch paths are usually used for things like special single player specific uses, or for enabling collectors to latch onto something to offload their resources.
3. Docking and incoming latch paths have to start off with a queue point and a first point far from a ship's avoidance boxes.
4. The last point of an incoming latch path is the same as the first point of an outgoing latch path.
5. Paths are represented by cone shaped markers.
6. Paths are created through the use of keyframes or 'points'. These are created when a point's values are highlighted, right clicked, and then 'Key Selected' is selected after necessary changes are made to said values. After this is done, the Time Slider is moved to a future point in time, values are adjusted if need be, and 'Key Selected' again.

11.8 Appendix – General Dock/Launch/Latch Path Information

Definitions of Path Parameters



There are two sections that define path parameters. The first section is shown in the picture with 'path17' (node parameters), and the other section shown on the left is labeled pathShape17 (shape parameters). Maya works with nodes and shape nodes. The parameters on the shape node (the ones covered in the section with pathShape17) are meant to be 'general' parameters, or parameters that don't change per path point. The node parameters on the other hand, are supposed to be animated. Maya shows a yellow background for parameters that have animation keys assigned to them, and a white background shows that the parameter has keyframes, and therefore is global for the whole path.



Shape Parameters

The Shape parameters are the parameters that define what the whole path is used for. Don't keyframe these parameters! These can be found in the path cone's Attribute Editor and some in the Channel Box when the path cone in question is selected.

Tolerance - Is obsolete and not used anymore.

ExitPath - Should be tagged if this path is used to launch ships. If it's used to bring ships in, this should not be tagged.

LatchPath - Should be tagged if this path is a latch path. *Latching* means that the ship doesn't go into a docking/hangar bay, but sticks itself to the outside of the docking ship (and remains visible for the duration of the docking operation).

Fighters...Transport - Enables this path for this specific docking family. This enables you to make certain docking paths accessible for a specific type of ship. Be careful though. **Utility** is for the resource collector, while the **Resource** parameter enables collectors to bring in containers for salvaging. The **Salvage** parameter is for the salvaging of objects in the game that aren't resource related (i.e. salvaging something in the single player campaign, for example).

UseAnimation - Means the use of the ships docking and launching animations to control the use of the path.

LinkedPathNames - A comma-separated list of path names. If any of the paths listed here are busy then this path counts as busy. This lets you link paths that use the same airspace so that only one can be in use.

Node Parameters

When a ship is flying along a launch/dock/latch path, it will use the parameters of the point it is flying towards. The different parameters can be set to a point by creating a key for the translation, and generating keys for all the other parameters.

Translate, Rotate, Scale, Visibility: Maya specific parameters. These will define the position and rotation of the docking/launch/path point. Don't scale or change the visibility for the launch/dock/latch path points.

Use Rotation: This specifies if the ship tries to match the rotation of this launch/dock/latch path point, or if the ship just tries to fly straight to the point. If you want the ship to turn a certain way, you can tag this. This will make the ship try to match the rotation. It might happen that the ship can't get to the exact rotation, but at least it will try. See 'Check Rotation' if you want to also *make sure* that the ship will reach the rotation.

Point Tolerance: Point tolerance is the tolerance the ship uses to mark this launch/dock/latch path point as 'done', and focus on the next point in the path. The tolerance is the distance in meters. If you want the ship to closely fly to this point, you want to set this to something small, like 20. If you don't really care if the ship is exactly on the spot, set this tolerance to something bigger, this will speed up the path, because it is less precise. Usually you set the point tolerance to lower values when the ship and the dock-ship are close to each other.

Drop Focus: This is the point where the camera loses focus if you were focused on a squadron that just docked with a target docking ship.

Max Speed: This is the maximum speed the ship can fly at this point in the path. If you want the ship to stop at a certain path-point, set this to 0. If you don't care about the speed of the ship, you could set this to something really high. The ships will never fly faster than their maximum speed. There is one exception: If this path is a launch path, the maximum speed of the first point in the path is used as the initial speed. This is done so that we can have fighters quickly launching out of ships, like torpedoes.

Check Rotation: Use this only if you really want to make sure the ship reaches the specified rotation. This is useful for launch/dock/latch paths of frigates, for example. If you want to rotate the frigate before entering the hangar, you want to make sure that the frigate is rotated correctly, before you move the frigate in. This is when you mark 'Check Rotation'. Usually a ship only checks the Point Tolerance to a launch/dock/latch path point, but if this parameter checked, also the rotation is checked before the ship continues with the next point.

Force Close Behaviour: Ships that are doing close behaviour basically slide (sideways, backwards, etc) to the target. If you want a frigate to move sideways along a path, you should turn on this parameter. If you use this, also use 'Use Rotation' - it would make no sense otherwise.

Player Is In Control: This is used in launch paths. The player can issue orders when the ships are launched a certain way along the path. Make sure **only one** point of a launch/dock/latch path is tagged with this. The game looks for the first point in the path that has this parameter checked. If the last ship in the squadron is launched, and has passed this point, the player can issue new commands. If the player does this, the ships will not finish their launch procedure, but they will obey the player immediately.

Queue Origin: Every docking path has a queue. The queue stacks ships up at the queue point. The first point of a *docking* path (so not an exit/launch path) should contain a queue origin flag to set the position and orientation of the queue. The second point in that launch/dock/latch path is the actual first point in the launch/dock/latch path.

Use Clip Plane: Only one point on the path should have this set. The game uses this point and the next one to define a clip plane that intersects this point. Anything before the plane will not be drawn. Use this when ships need to spawn behind a wall but there's no space back there to hold the ship. By default all paths have the first point set as the clip plane. Use this to change the plane to a further point. An example of this can be found in the super capital ship launch path of the Hgn_Shipyard.

Clear Reservation: Clears the path reservation so that the next ship to use the path can start its operation.

Animating points in Maya

To animate the cones in Maya, look at the bar at the bottom of the screen (picture on the right).

The red lines are the keyframes for the currently selected cone.

The exporter looks at keys on the **translation** parameter of the cone, so make sure that you key those every time you want to create a point.

Create a keyframe by dragging the time slider to the frame you want to create the key at. Then move the cone parameters to the right value. Then, after shift selecting the parameters you want to create animation keys for, right click and hold them. This brings up a menu. Select 'Key Selected' to create the key. **Remember to only create keys for the node parameters, not the shape parameters!**

One really important thing to remember is to **NEVER create keyframes at frame 0! Start at frame 1!** This is because of exporter problems a long time ago.

Avoidance, Collision, Paths, and You!

The avoidance and collision systems can be annoying when you are creating launch/dock/latch paths. If you are having problems with these, it probably is because you need to alter your settings a little so that the avoidance and collision systems don't get frustrated with the position of the ships.

The avoidance and collision systems are there to prevent ships from ramming into each other. This means they make sure that there is a considerable distance between all the ships. The distance is minimized already to make sure there can't be a collision under 90% of the circumstances. Even if the avoidance box looks too big, there is always a reason why it is that big.

One important thing to know and that we can't stress enough is that:

The starting point of a docking path needs to be outside the avoidance box of the dock-ship for that type of ship.

However, avoidance boxes don't have a set size. They resize dynamically to match the sizes of both ships that are avoiding each other.

For example, say a resource collector is trying to avoid a mothership. The avoidance box of the mothership is defined by the collision box of the mothership plus the width, height, and depth of the resource collector, plus a small distance, and plus a factor of the velocity of both ships. The small distance is there to have the ships not clip each other. This way, we only need to check the midpoint of the collector to see if it needs to find a path around the mothership. This will result in an avoidance box that is slightly bigger than the mothership's collision box.

Now, imagine a battlecruiser that is trying to avoid a mothership. The battlecruiser is a lot bigger than the resource collector. This means that the avoidance box of the mothership is a lot bigger as well. The midpoint of the battlecruiser needs to be a lot further away to avoid clipping of the two ships.

Hopefully this makes the point of why the docking paths for certain ships need to be further out than others.

12 Optimizing

Source HW2 Ship files can be very complex and contain a myriad of 3D geometry, joints and textures. This section details the steps required for finalization of all artwork in preparation for exporting to HOD files (Homeworld Object Data).

12.1 Optimization Techniques

The most important thing to keep in your mind when you make a ship is to **minimize the number of state changes**! Do not use thousands of triangles. The two causes for state changes are different **shapes** or different **materials**. You can go ahead and keep your ship made out of dozens of shapes while you work on it for your convenience, but be sure to merge them all together if possible at the end. The only reason I can think of for having multiple shapes is because they are on different joints that need to animate.

The other thing that causes state changes is different materials. Try your best to use as few materials as possible per ship. The easiest way to do that is to use one giant texture instead of multiple small textures.

Ideally there would be only a single material and no joints. While you can't do this very often, it should be able to happen at lower LODs.

Also **don't make low resolution copies of textures**! If you have a 256x256 texture you automatically have all the mip map levels for it. That means if you have a 256x256 texture you also have a 128x128, 64x64, 32x32, 16x16, 8x8, 4x4, 2x2, and 1x1. Do **not** go into Photoshop and make a 128x128 version and 64x64 version for other LODS, just use the existing 256x256 version and the system automatically uses the correct mipmap level. Never have a high and low res version of the same texture. **Only make low resolution copies of textures if your lowest LOD models demand it (due to unavoidable texture warping)**.

To reduce texture memory usage, there is one very simple trick that has a huge effect that you can do to help get under budget: **Use smaller glow maps**.

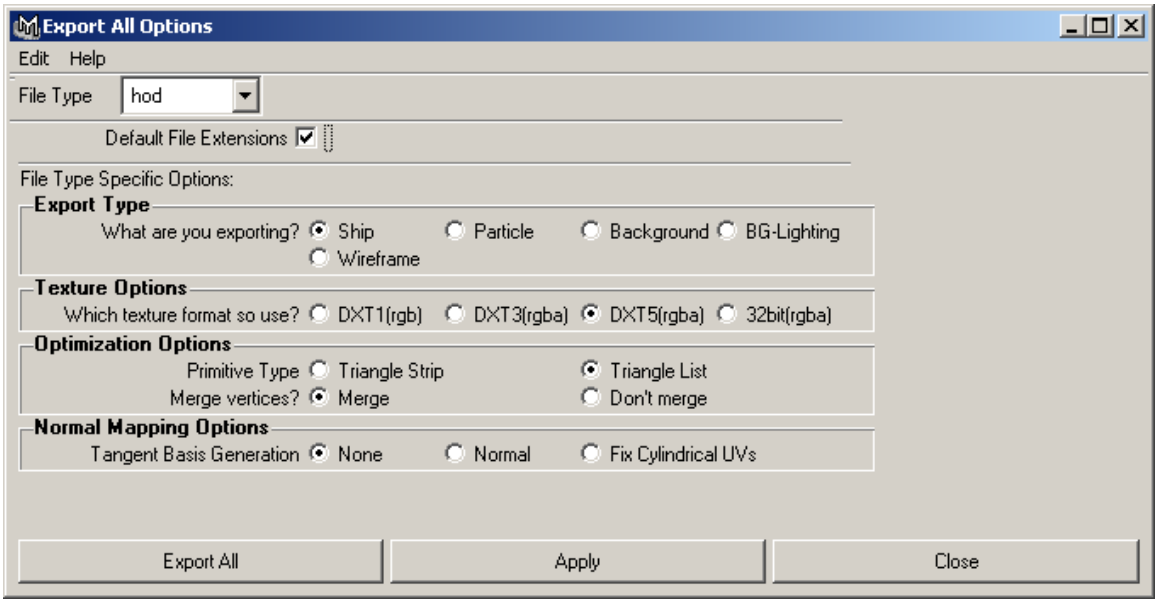
When you get your **$glow** layer in your **export** texture try adding a /2, /4, /8 or even lower suffixes, depending on what you deem tolerable. i.e. your two layers will be **$diffuse** and **$glow/4**. In many cases you can go much lower. In some cases you may have no **team stripe**, no **glow**, and no **specular**-just pure black. So a 1x1 texture is as good as a 64x64 texture. In that case add **/64** and your glow layer will be resized to a 1x1 texture.

In Maya, turn on **back face culling**. If you don't do this you'll be able to see both front and back facing triangles in Maya. In the game you only see front facing triangles.

13 Exporting

The basic steps for exporting any ship are as follows:

1. Load the model into Maya
2. Make sure you select `Edit → Delete All by Type → History` to delete unnecessary history that may clutter up the file. Save your work.
3. Click on the box beside `File → Export All`. This will open the `Export All Options` window:



4. Beside where it says `File Type`, there should be a drop box. Click on it, and select **hod** from the list.
5. Make sure that the **`Ship`** radio button is selected in the 'Export Type' section.
6. Make sure that 'DXT5(rgba)' is selected in 'Texture Options.'
7. Make sure that in the 'Optimization Options' that Triangle List is selected and Merge is selected.
8. Once this is done, click on the 'Export All' button.
9. Select the directory the .hod is going into. In the case of our hypothetical ship, we will place its .hod in 'Homeworld2/Data/Ship/<name of ship>'. Thus, we will export our Hgn_Prototype.hod file into 'Homeworld2/Data/Ship/Hgn_Prototype'.

The following is a brief explanation for the different export options.

File Type

The file type must be set to 'hod' for when exporting models.

Export Type

While five export types are available, only four of the five are actively in use.

- Use the 'Ship' option for ships, subsystems, asteroids, and resources (dust clouds, nebulae debris).
- Use the 'Particle' option for most Advanced Tactical Interface (ATI) graphics including state icons and health bars. Also use the 'Particle' option or non-wireframe 3D UI elements such as the direction plate and bearing labels in the sensors manager.
- Use the 'Wireframe' option for wireframe elements in the ATI.
- Use the 'BG-Lighting' option for background lighting.

Texture Options

3 DXTx compression options and an uncompressed option are available. By default, DXT5 is used for all export that involves textures (Ship and Particle export). The texture option is irrelevant when exporting background lighting and wireframe models since no textures are exported.

Optimization Options

The 'Triangle List' and 'Merge' options are recommended for all the export of all Homeworld2 models as they offer the best performance in the tests we have conducted. We suspect that triangle lists out perform triangle strips due to the faceted nature of most models. Merging shared vertices simply reduces the number of duplicated vertices.

Normal Mapping Options

Use the 'None' option for models that do not have normal maps. Use the 'Fix Cylindrical UVs' option for asteroids. This option is needed for asteroids since the texture wraps around and special treatment is needed where the mesh seams up.

14 Ship Tuning

This section details how to get a ship into the game once the appropriate game assets have been created.

14.1 Getting a Ship Into the Game

1. Open ship tuning, Enable macros
2. Create a new column
3. Copy the column of a similar ship; For example if you want a fighter class ship copy a fighter column
4. Paste the copied column into your previously empty column
5. Rename the column to what you want the ship to be called
6. Create\Update\Export Ship Data (The grey button at the top left of shiptuning.xls)
    a. This will create a .ship file in Homeworld2\Data\Ship\<name of ship>\<name of ship>.ship
        i. NOTE: When exporting a ship file Hiigaran will be shortened to HGN and Vaygr to VGR
    b. Note that the name of the .ship file needs to be **identical** to the subdirectory name.
7. Add the appropriate HOD file to Homeworld2\Data\Ship\<**name of ship**>\<**name of ship**>.hod
    a. Note that the names of the .hod file and the .ship file need to be **identical** to the subdirectory name.
8. Add the EVENT file to Homeworld2\Data\Ship\<name of ship>\<name of ship>.event
9. To make the ship available in game Open Data\Scripts\Building and Research
    a. Open up the race directory you want to add to and then build.lua
    b. For more information on adding a ship to build.lua read HW2_ResearchandBuidlingScripting.doc

14.2 Art Related Items Changeable in Shiptuning

The following items can be manipulated in shiptuning.xls in some form or another. A brief explanation of what you can change follows. Links to in-depth explanations of how to create the assets to hookup in shiptuning.xls accompany the brief descriptions.

1. UI and text displayed to the user
    i. The ship name is Located at the top of shiptuning.xls and displayed in the bottom UI bar when in HW2. This needs to match the name of the ship that you using in build.lua, otherwise your ship will have two different names.

ii. SOB Description is located at the top of shiptuning.xls just underneath the ship name. It display's the Unit Role, located in the bottom HW2 UI bar.

2. Adding a new weapon to a ship
   i. Create a new weapon in weapontuning.xls (this is described under Weapons).
   ii. Under the Weapons section in shiptuning.xls place the <weapon name> of a weapon you have created
   iii. Now put in the Joint name found in the MA file. For information on creating a joint click here. This is essentially attaching the weapons stats that you created in weapontuning.xls to the joint found on your ship. That joint will now shoot the weapon you created.
   iv. Add a weapon firing effect to the Fire Event line. See HW2_FX_Tool_Documentation1.doc for how to create a weapon effect.
3. NLIPS
   i. These values are used to make objects appear larger then their actual size when the camera is far away from them. It is useful in the fact that it makes it easier to identify objects when they are far away. There is a starting distance when NLIP begin (the ship starts to look bigger then it actually is) and it linearly scales until it reaches the NLIP Far Range at which point the ship does not increase in size anymore.
4. Subsystems
   i. The ability to define the subsystems available on a per ship basis is done with ShipTuning.xls under the subsection "Hardpoints"
   ii. So for our subsystem to show up, type 'Hardpoint' in the jointname box, 'Production' in the Type box, 'Indestructable' in the Healthtype box and 'Hgn_TestSubSystem' in the DefaultSubSystem box. Now if you load the ship in the game, the subsystem should show up. For more information, read subsystems and shiptuning.
   iii. For more information on creating subsystems please reference the subsystem section.
5. Capture and Repair points
   i. Enable or disable Capture under "Can Be Captured" in shiptuning.xls
   ii. Enable or disable Repair points under "Can Be Repaired" in shiptuning.xls
   iii. For more information on creating Capture and Repair points refer to that section

6. Level of detail (LOD)
    i. Toggle the distances you want the LOD's to kick in
    ii. For more information on creating LOD's refer to that section
7. Ship Icons
    i. The file to hook up ship icons is located in: C:\projects\Homeworld2\data\Ship\Icons\ShipIcons.lua
    ii. Refer to the Ship Icons section for more information on creating and hooking them up
8. Docking paths
    i. To allow a ship to dock open shiptuning.xls, ensure that "dock command" field is filled out. Ability active by default should be enabled along with dock time between formations etc.
    ii. Under families in shiptuning.xls make sure an appropriate dock family is added. This should be chosen based on size of the unit. Fighters sized units should use the fighter dock family while frigate sized units should use the frigate dock family.
    iii. For information on creating docking paths read this section.

15 Appendices

15.1 Bugfix/Troubleshooting FAQ

My ship won't export

Make sure the mesh is parented under a Root joint.

Maya crashes when I attempt to refresh a PSD texture file

PSDs can have images outside of their visible canvas area. Maya cannot accept such files. Select the entire image and crop to the canvas extents, save, and try to refresh the image in Maya again.

My ship's Weapons/Subsystems/Capture & Repair Points/FX are not firing/spawning/working/triggering

In Maya, make sure the joint names for the ship element in question correspond the to their xls file equivalents.

15.2 Ship Rescaling

Throughout the course of art production it may occur that relative ship scales may need to be tweaked (heaven forbid!). If and when this occurs it is crucial to use the following process to rescale ship sizes because incorrect rescaling can lead to unwanted loss of Maya shipfile data.

Make sure `FreezeShipScale.mll` is loaded and is autoloaded in Maya.

To freeze the ship's scale:

1. Select all the goblins.
2. Freeze transform.
3. If any of the goblin's normals are reversed then:
    a. Select one of the goblins with reversed normals
    b. Reverse the face normals
    c. Reload the ship (Note: You don't want to restart Maya. Reloading the ship after reversing the face normal somehow puts Maya in the proper mode.)
4. Select the Root joint.
5. Scale it (isotropic scaling recommended).

Hit the `Freeze Ship Scale` button in the Relic tool shelf.

15.3 New User Interface Document

The following is taken from the NewUserInterface.doc.

Input events are received from the Homeworld2 region code. Since all screens a background region that tie into this system, a callback is setup to receive mouse and key input events. Once a mouse or key event is captured in this callback, the ScreenManager is called to delegate the event to the necessary screen, which will then delegate it to the proper interface elements.

Interface Elements

Standard Element	***Description***
Button	A 4-state button that is for buttons, toggle buttons and checkboxes. The states are Default, Over, Clicked, and Pressed. When the mouse is pressed on a button in its Pressed state, it will change to its clicked state.
TextButton	An extension of the Button class that has a text field.
TextLabel	A simple text label widget. Supports text wrapping and newline characters. (“\n”)
Frame	An empty element capable of capturing events. Has a background color, and / or can contain a background graphic.
ProgressBar	Progress bar has a min and a max range. The progress can be set using a value within or the range or via a floating point number with range [0.0,1.0] The progress bar background can be an RGBA color or a graphic. If a graphic is used, the texture will be clipped to the progress rectangle, instead of stretched.
ScrollBar	Horizontal and Vertical scrollbar. To use scrollbars you must define the up/down right/left arrows as well as the track bar and the background frame. The trackbar will resize depending on the range of the scrollbar (similar to recent versions of windows)
Table	A multi-cell table widget. Has column/row titles and cells.
TextInput	Single-line text input box. Emits ‘text changed’ signals.
ListBox	Displays a scrollable list of list box items.
DropDownListBox	Just like windows drop down combo-box.

Dependencies

The UI System depends on the following files:

- ***UISettings.lua*** – Contains stylesheets, and tables of containing all loaded UI screens.
- ***UISound.lua*** – Contains sound hookups
- ***AStyleSheet.lua*** – You will need a stylesheet. You can name this whatever you want.
- ***UIStringTable.lua*** – Contains a table of localized text strings indexed by name. The UIStringTable singleton is used in code for static access to localized strings.

The UIScreen

The UIScreen contains a root element (a Frame element) which acts as the container object for the screens elements. So when you call UIScreen::AddElement you are really adding a child to the frame root element. Typically, in your user interface you will have a couple of different UIScreens that you will show, hide, and move around. UIScreen management such as loading, activating, and unloading is done through the UIScreenManager singleton.

The UIScreen receives updates once per frame. On update, the screen will check all input and deliver mouse and key events to affected elements. The UIScreen also handles special cases such as focused elements (only the focused element will receive key input), input locked elements (example: when you click the trackbar of a scroll bar, if you move off of the scrollbar the element will still receive mouse events until you release the mouse button.)

Regions

Since the Homeworld2 region code plays an important role in the event based UI, I thought it might be worth mentioning. When a screen gets made visible, a region is created that has the same position and size as the screen. Ensures that when you are in game, playing HW2 and a UIScreen is clicked, the event will go to the screen and no the underlying game.

Clearly, there is some duplication in the region code and the UI input handling code; however, we chose to implement separate input handling code for the UI so it wasn't totally dependent on the HW2 regions. The most important factor in the interaction of the region and UI code is that every time an input event occurs (mouse or keyboard) the UIScreenManager is notified via the HandleEvent(UI::EventType type, UI::EventInfo info) method.

Events & Event Handling

The UIScreen will be responsible for propagation of the following events to any affected interface elements as well as any of their affected children:

MousePressed, KeyPressed	Mouse or key has just been pressed over element. For an element to receive key events, it must have the focus.
MouseReleased, KeyReleased	Mouse or key has just been released over element. For an element to receive key events, it must have the focus.
KeyRepeat	Key repeat events will be sent to an element when a key is held down for a long period of time.
MouseDoubleClicked	Mouse was double clicked on element within a set time period.
MouseClicked	Mouse was released on the same element it was pressed on.
MouseOver	Mouse is over the element; check the button state to see if the user is holding a mouse button down.
MouseEnter	The mouse has just entered the element; check the button state to see if the user is holding a mouse button down.
MouseExit	The mouse has just left the element; check the button state to see if the user is holding a mouse button down.

When an interface element receives an event it will emit its signal (*see signals and slots)* for that event, which will trigger any connected slots connected to that event. It will also call protected event handlers that can be overridden by any element subclass:

Overridable event handlers:

```
protected:
  virtual void OnMousePressed(Vector pos, Button b)
  virtual void OnMouseReleased(Vector pos, Button b)
  // and so on…
```

This provides us with an easy way to capture events in derived classes. For example a button element would look something like this:

```
class Button : public InterfaceElement {
public:
 Button(...)
 // so on

protected:
 ButtonState m_state;

 virtual void OnMouseEnter(Vector pos, Button b){
   m_state = BS_MOUSEOVER;
 }
 virtual void OnMouseExit(Vector pos, Button b){
   m_state = BS_DEFAULT;
 }

};
```

Because the event handlers are virtual methods from InterfaceElement, the Button class will receive the events instead.

UIScreenManager (SINGLETON)

The UIScreenManager contains a list of all loaded and all active UIScreens. A loaded screen is loaded into memory and does not receive draw or update events. When you want a screen to receive updates you can tell the UIScreenManager to activate that screen.

The UIScreenManager will be tied into the game engine to receive updates once per frame. On update, the screen manager will call draw and update methods for all of its active UIScreens.

Signals and Slots

Signal and slots will be used for the binding of callbacks to GUI actions. For example, every time a button is clicked a click signal will be triggered. When a signal is triggered, any slots that are connected to it will be called. This allows us to connect multiple callback functions to GUI events: very useful.

The InterfaceElement has a signal for every type of event that it handles. This allows us bind multiple callback methods to any event (MousePressed, MouseReleased, KeyRepeat, MouseEnter etc.)

The basic elements (Button, ListBox, ScrollBar) also have some custom signals that are called on button click, on scroll, on list box item select etc.

When writing your own custom elements, you may wish to use your own custom signals. See Signals.h for more documentation on using signals and slots.

An example:

```
void MyButtonClicked(InterfaceElement * sender)
{
        // The sender is the element that called the signal.
        // Note: You may want to do a dynamic cast here if you are not sure on the type.
        Button * button = static_cast<Button *>(sender);

        // Put your code here to execute every time the button is clicked
}

void MyButtonMouseExit(InterfaceElement * sender)
{
        // Put your code here to execute every time the mouse exits the button.
}

void SomeWhereInCode( )
{
        Button * myButton = new Button(parent, "myButtonName");
        myButton->ConnectOnButtonClicked(MyButtonClicked);
        myButton->ConnectOnMouseExit(MyButtonMouseExit);
}
```

LUAConfig and Loading UIScreens

UIScreens and their elements will be created through LuaConfig script. To create a new UIScreen you will need to create a .lua file for the screen def'n and add it to UISettings.lua. If you need additional code hookups (which most screens will) you will have to create a cpp class for the screen and add it to the menu factory in MenuFactory.cpp. See UIScreenWalkthrough.doc for a step-by-step walkthrough to creating a UI screen.

The biggest potential problem with the loading the screens from lua script is that to represent a complex control (such as a scroll bar) with a Lua script you will need hundreds of lines of script everytime you need to place a scrollbar. To solve this problem we use stylesheets.

Stylesheets

Stylesheets allow you to create an element that uses a pre-defined style for its default properties. Once the elements properties have all been set to that of the stylesheet you can then 'override' any additional or existing attributes. For example, say you want to create two text labels that share the same font and size, but one is bold and one is italic.

The way to do this with stylesheets would be to create a text style with you font, your size, and bold typeface. When you create your first text label, you will only need to specify the text style and when you create your second text label you will need to specify the text style as well as the italic typeface. This, in effect, will override the typeface of the text style.

Example:

```
-- In your stylesheet file
MyStyleSheet = {
	MyTextStyle = {
		type = "TextStyle",
		font = "Arial",
		fontSize = 8,
		typeFace = "Bold",
	},
},

-- In your UIScreen file
MyUIScreen = {
	stylesheet = "MyStyleSheet",

	-- Create a bold text label with the same attributes as the text style
	{
		type = "TextLabel",
		name = "MyBoldTextLabel",
		textStyle = "MyTextStyle",

		position = {0, 0),
	},

	-- Create an italic text label by overriding the style sheet
	{
		type = "TextLabel",
		name = "MyBoldTextLabel",
		textStyle = "MyTextStyle",

		position = {0, 0),
		typeFace = "Italic", -- Stylesheet overide
	},
},
```

The way that stylesheets are handled internally is that each InterfaceElement is given a local copy of the element style and it is initialized from the linked style or by the default style if no linked style is given. Once the local style is initialized, any later changes will overwrite the attributes of the local style, thus overriding the linked or default style.

For a more detailed look at stylesheets see Stylesheets.doc.

Multi-Res Textures

Support has been added for multiple resolution textures. Multi-res textures are similar to mipmaps as they allow higher detailed textures to be used at higher resolutions. When specifying texture filenames in the Graphics sections of elements you can pass in a multi-res texture (*.mres). An mres file is simply a text file specifying a set of textures to use at certain resolutions. An example mres file looks like this:

```
baseRes = 800

res800 = "DATA:/UI/NewUI/Taskbar/background800.tga"

res1024 = "DATA:/UI/NewUI/Taskbar/background1024.tga"

res1280 = "DATA:/UI/NewUI/Taskbar/background1280.tga"

res1600 = "DATA:/UI/NewUI/Taskbar/background1600.tga"
```

The baseRes represents the resolution in which the UV coordinates are specified if they are in pixel space. In HW2 we always used pixel space UV coords since it was all hand scripted. Having to convert between the two in your head all the time was too error prone and unnecessary.

When using multi-res textures, you do not explicitly need to define a graphic for every resolution. For example, just having a res800 would be fine. That texture would just be used at every resolution.

LUAUserInterface Lib

LuaUserInterface is a lua lib defined in LuaUserInterface.cpp. These are lua bound functions that can be called from UI screens as well as the SCAR system. There is room for expansion in this area. This library is far from complete as we added functions only as we needed them.

The current functions are documented with doxygen in

homeworld2/documents/technical/chapters/ui/luauserinterface/luauserinterface.html

Sound Hookup

The UI system runs all of its sound through the UISoundManager singleton. This isolates the dependings on the sound system and provides an easy way creating global UI sound settings such as UI Volume.

Heres a snippet of the UISound.lua file from Homeworld 2:

```
feSoundRoot = "data:sound/sfx/ui/frontend/"


-- All buttons have a click sound

SFX_ButtonClick = {

        filename = feSoundRoot.."Mouseexitbuttonstyle2",

}


-- All Menu and NavBar (Back, Start, Ok, Cancel) have mouse enter

SFX_ButtonEnter = {

        filename = feSoundRoot.."Mouseoverbuttonstyle1",

}
```

```
-- Tabs (eg. in the options menu) have an enter, but no click
SFX_TabEnter = {
        filename = SFX_ButtonEnter.filename,
}


SFX_ChatMessageReceived = {
        filename = feSoundRoot.."Chatmessagereceived",
}
```

These filenames are then passed on to the sound system. The HW2 sound system never required a file extension for the sound files thus the filenames with no extensions.

These sounds could then be played from the script function UI_PlaySound(const char * soundname) bound in LuaUserInterface or via the custom sound attributes for the widgets. For example, the InterfaceElement class has strings containing sounds to play on events that could be specified in the script files.

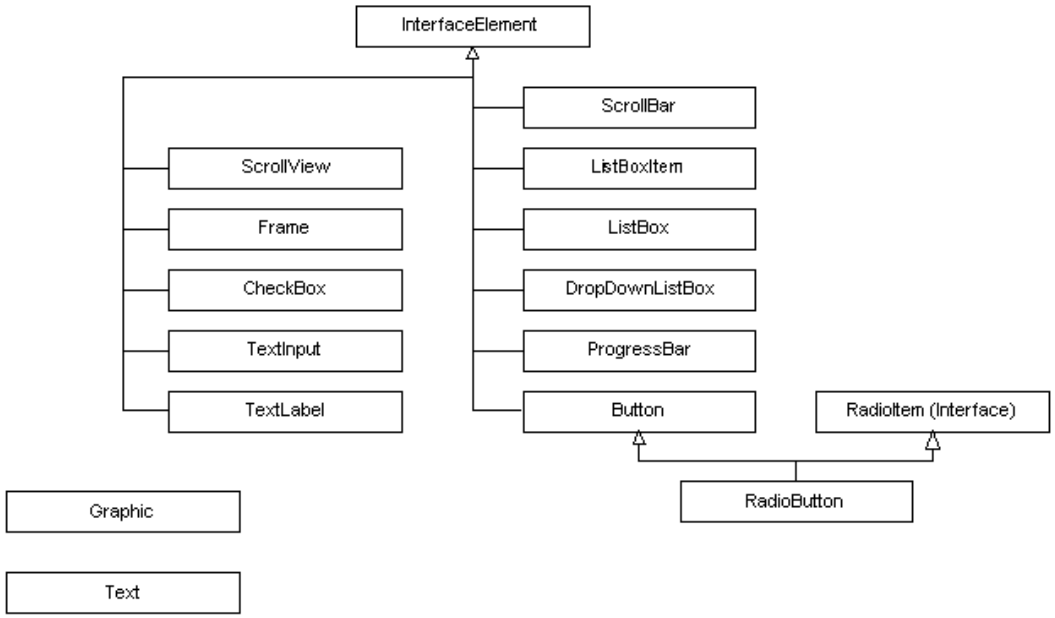
```
{
        type = "Frame",
        name = "frmFunkyBeats",

        -- sound hookup
        soundOnEnter = "SFX_ButtonEnter",
        soundOnExit = "SFX_ChatMessageReceived",
        soundOnClicked = "SFX_ButtonClicked",
        soundOnPressed = "SFX_ChatMessageRecieved",
        soundOnReleased = "SFX_ButtonEnter",
},
```

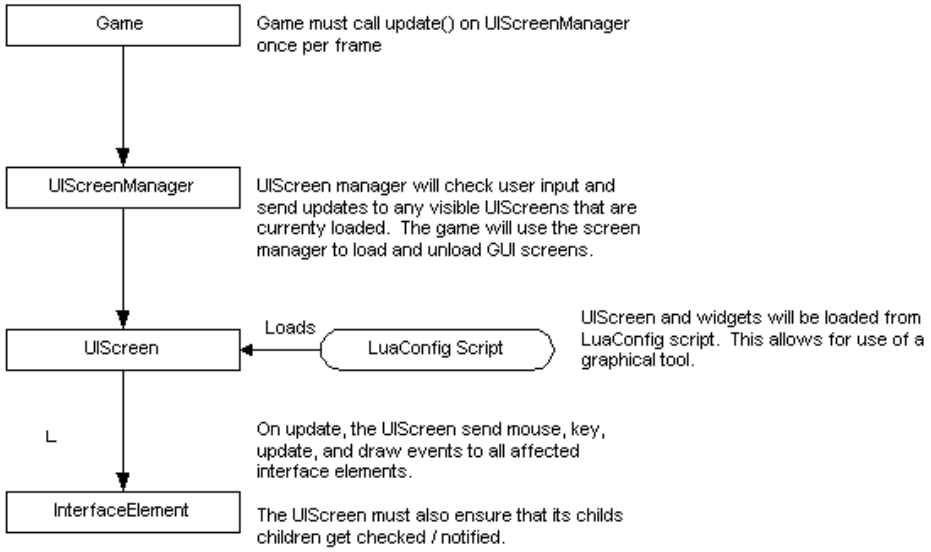
Almost all sound hookups for widgets were done in the style sheets. This is due to consistent nature of UI sounds. It was only in rare cases where we needed to put sound hookups in the screen definition files.

Diagrams/Reference

Interace Elements (Widgets)

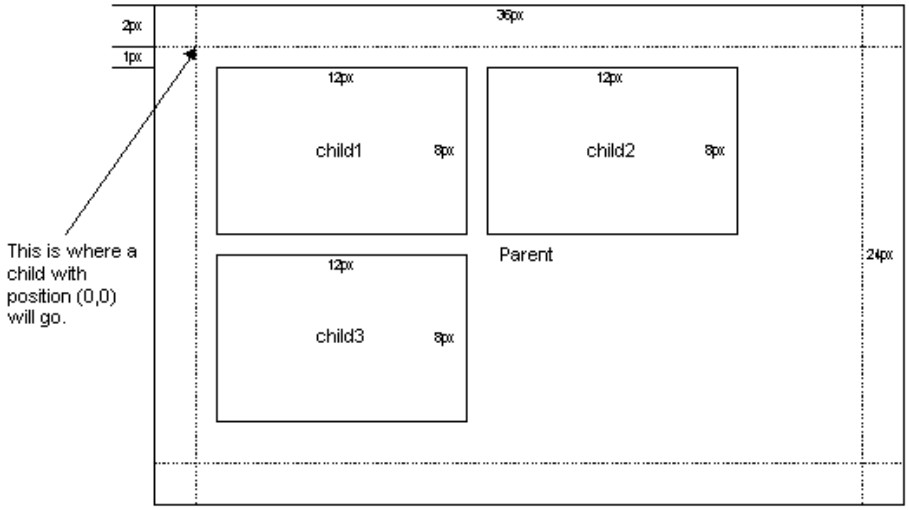


UI System



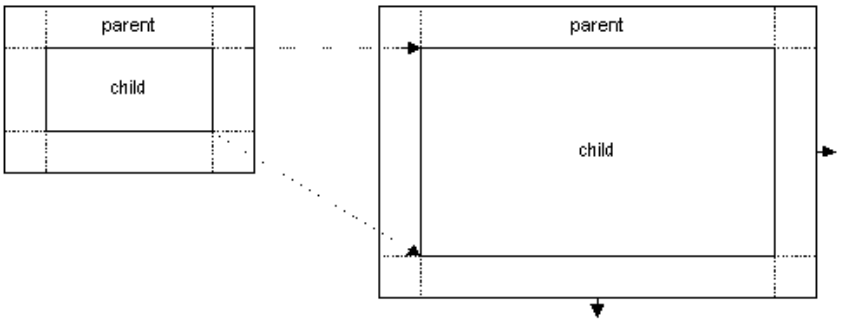
Parent with autoarranged children and margins

Demonstrates a parent widget with three autoarranged children. The parent has a margin width and height of 2 and autoarrangeSpace is set to 1. When child3 gets added it will detect that there is not enough room in the parent widget to fit, and it will wrap.



Parent that autosizes to children

Demonstrates a parent with the autosize flag set. The parent widget has a margin width and height of 2. If the parent has more then one child it will resize to their greatest bounds.



Children that resize to parent

Demonstrates a child widget with the resizeToParent flag set. The parent widget has a margin width and height of 2. All children with the resizeToParent flag set will get resized. The childs position will never change, resizeToParent only affects the width and height.