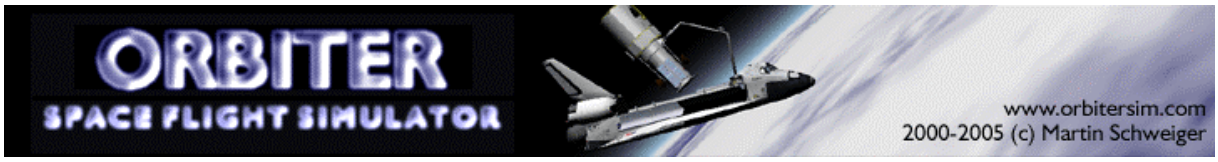
ORBITER
Guide du Développeur : Modèles 3D

Site internet d'Orbiter : orbit.medphys.ucl.ac.uk/ ou www.orbitersim.com



Contenu

1 Introduction

Ce document contient des directives de développements de nouveaux modèles 3D pour la visualisation de vaisseaux spatiaux dans Orbiter. Il concerne les développeurs d'addons, ou quiconque souhaitant étendre la fonctionnalité d'Orbiter.

2 Comment créer une nouvelle classe de vaisseaux spatiaux

Pour ajouter une nouvelle classe de vaisseaux spatiaux à ORBITER, les étapes suivantes doivent êtres réalisées :

- Définir les *paramètres physiques* de la nouvelle classe de vaisseau spatial dans un fichier de configuration dans le sous-répertoire `Config`.
- Créer un *surface mesh* qui défini l'apparence visuelle du vaisseau, dans le sous-répertoire `Meshes`.
- Optionnellement, ajouter n'importe quelles textures utilisées par le vaisseau, dans le sous-répertoire `Textures`.
- Ajouter un *scenario* qui inclut un vaisseau (ou plus) de nouvelle classe dans le sous-répertoire `Scenarios`.

Les étapes ci-dessus vous permettent de créer un vaisseau basique avec des paramètres génériques.
Pour personnaliser entièrement votre nouveau vaisseau, une étape supplémentaire est requise :

- Ajouter un module DLL pour la nouvelle classe de vaisseau qui personnalise son comportement (parties en mouvements, personnaliser les panneaux du cockpit, personnaliser le modèle de vol, etc.) dans le sous-répertoire `Modules`.

Se référer au document OrbiterAPI pour connaitre les informations nécessaires à l'écriture de modules de vaisseaux pour Orbiter. Un certain nombre d'exemples de modules avec le code source sont contenus dans le dossier Orbitersdk\samples.

Notez que chaque classe de vaisseau spatial *doit* avoir un fichier de configuration, même si tous ses paramètres sont définis dans un module DLL. (Dans ce cas, la seule entrée dans le fichier de configuration peut être le nom de module.)

3 Le fichier de configuration de classe de vaisseau

Les fichiers de configuration sont des fichiers textes ASCII qui peuvent êtres édités avec n'importe quel éditeur de texte étant capable d'écrire de simples fichiers textes (ex. `notepad`). Tous les fichiers de configuration de vaisseau se trouvent par défaut dans le sous-répertoire `Config` d'ORBITER (A moins que l'entrée *ConfigDir* dans `Orbiter.cfg` n'apparaisse dans un autre dossier).

Ci-dessous, une description d'options de configurations par défaut de vaisseau reconnue par Orbiter.
Notez que toutes les options n'ont pas besoin d'êtres présentes dans un fichier de configuration.
Certains vaisseaux particuliers, définis via des modules personnalisés, peuvent spécifier divers paramètres directement dans le module. De plus, les modules de vaisseau peuvent lire des paramètres personnels additionnels non-affichés ici depuis le fichier de configuration.

Item	Type	Description
BaseClass	S	Optionnel; classe parent. Les entrées recherchées sont issues de cette classe. Permet la construction de hiérarchies de classe. (Soyez sûr de ne pas introduire des dépendances circulaires !)
Module	S	Optionnel; nom du plugin de module pour la personnalisation du vaisseau. Le module doit se trouver dans le dossier *Modules*.
Help	S,S	Optionnel; nom du fichier help utilisé pour l'aide spécifique de classe de vaisseau lorsque l'utilisateur presse le bouton "Vessel" dans la fenêtre "Help dialog". Le fichier help doit être un fichier html compilé (.CHM) et localisé dans le répertoire Html/Vessels. L'entrée contient le nom de fichier sans chemin et extension, et (séparé par une virgule) le nom de la première page du fichier à afficher (sans extension). Par défaut : no vessel class specific help.
EditorCreate	B	Si *false*, le type de vaisseau n'apparait pas dans la liste de la page de création de vaisseau de l'éditeur de scénario. (Par défaut : *true*)

ImageBmp	S	Nom du fichier bitmap (BMP) qui affiche le vaisseau. Le nom doit inclure le chemin (relatif au répertoire principal d'Orbiter) et l'extension (.bmp). Cette image est affichée sur la page de création de vaisseau de l'éditeur de scénario. Pour de meilleurs résultats, les dimensions devraient êtres de 164x240 pixels.
MeshName	S	Nom du mesh utilisé pour la visualisation.
EnableFocus	B	*true* si le vaisseau peut recevoir une mise au point d'entrée (défaut: *true*)
EnableXPDR	B	*true* si le vaisseau a un transpondeur (par défaut: *false*)
XPDR	I	Canal du transpondeur (en unités de 0.05 kHz à partir de 108.0 kHz). Utilisé seulement si EnableXPDR=true. Ce canal par défaut peut être redéfini par un script de scénario de vaisseau.
Mass	F	Masse du vaisseau (vide) [kg]
Size	F	Rayon (moyen) du vaisseau [m]
MaxMainThrust	F	Taux de poussée des moteurs principaux [N]
MaxRetroTrust	F	Taux de poussée des moteurs de retro-poussée [N]
MaxHoverThrust	F	Taux de poussée des moteurs de sustentation [N]
MaxAttitudeThrust	F	Contrôle du taux de poussée des moteurs à réaction [N]
TouchdownPoints	V V V	3 points de contact avec la surface dans les coordonnées locales du vaisseau. Pour des configurations de type avion se sont : La roue du nez de l'avion, la roue principale gauche, la roue principale droite. (L'ordre est important pour définir la direction "up"). D'autres types de vaisseaux spatiaux peuvent interpréter les points différemment.
CameraOffset	V	Position de la caméra à l'intérieur du vaisseau pour la vue cockpit.
CW	F F F F	Coefficients de résistance du flux d'air : devant, derrière, transversal, vertical. Utilisé uniquement par le modèle de vol hérité (Si le module ne définit pas de profils aérodynamiques).
WingAspect	F	L'aspect des proportions de l'aile (wingspan2 / wing area). Utilisé pour le calcul de la trainée atmosphérique dans le modèle de vol hérité.
WingEffectiveness	F	Un facteur de forme d'aile : ~3.1 pour des ailes elliptiques, ~2.8 pour des ailes effilées, ~2.5 pour des ailes rectangulaires. Uniquement utilisé pour le modèle de vol hérité.
CrossSections	V	Sections transversales dans les directions des axes (z=longitudinal) [m2]
RotResistance	V	Résistance contre la rotation autour d'axes dans l'atmosphère, où la décélération angulaire due à la friction atmosphérique est de $a(\omega)x,y,z = -v(\omega)x,y,z \rho r x,y,z$ avec une vélocité angulaire $v(\omega)$ et une densité atmosphérique ρ.
Inertia	V	Principaux moments d'inertie, masse-normalisés (voir ci-dessous) [m2]
GravityGradientDamping	F	Coefficient d'amortissement pour « gravity gradient torque ». Détermine le temps de relaxation pour « tidal locking ». Par défaut: 0 (non amorti)
PropellantResource*i*	F[F]	Estimations de réserve de carburant *i* (*i* ≥=1). Première valeur : max.fuel capacity [kg]. Seconde valeur : fuel efficiency factor (>0, par défaut: 1)
MaxFuel	F	Masse maximum de fuel [kg]. Obsolète ; Utilisé seulement si les réserves de carburant ne sont pas définies.
Isp	F	Valeur par défaut de l'impulsion spécifique du carburant [m/s]: Quantité de poussée [N] obtenue en brulant 1kg de carburant/seconde. Les modules de vaisseau peuvent redéfinir cette valeur pour des moteurs individuels.
MEngineRef*i*	V	Position de référence des moteurs principaux *i* (*i*=1…)
REngineRef*i*	V	Position de référence des moteurs de retro-poussée *i* (*i*=1…)
HEngineRef*i*	V	Position de référence des moteurs de sustentation *i* (*i*=1…)
AttRef*dij*	V	Position de référence des moteurs d'attitudes (pour une rotation autour de l'axe *d* (*d*=X,Y,Z), direction de rotation *i* (*i*=1,2) et index *j des* moteurs (*j*=1,2)) pour un total de 12 moteurs d'attitudes.
LongAttRef*ij*	V	Position de référence des moteurs d'attitudes (pour un mouvement de translation linéaire avant/arrière), direction *i* (*i*=1,2) et l'index de moteur *j* (*j*=1,2)) pour un total de 4 moteurs d'attitudes.
DockRef	V	Point de référence d'amarrage du premier port d'amarrage. (obsolète)
DockDir	V	Direction d'approche d'amarrage du premier port d'amarrage. (obsolète)
DockRot	V	Direction d'alignement longitudinal (normal to DockDir) pour le premier port d'amarrage (obsolète)
<Docklist>	List	Liste des positions et des directions d'approches des ports d'amarrage (voir ci-dessous).
<Attachment list>	List	Liste des positions et des directions d'approches des points d'attachements (voir ci-dessous).

(S=String, B=Bool, F=Float, V=Vector)

Notes:

- Une classe de vaisseau peut être issue d'une classe de vaisseau différente, en définissant l'entrée BaseClass. Toutes les propriétés non définies dans le fichier de configuration de la nouvelle classe sont prises de la classe de base.
- Le nom du mesh ne doit pas contenir l'extension de fichier (.msh) et ne doit pas contenir de chemin de répertoire.
- L'entrée MaxFuel a été remplacée par PropellantResource, qui permet la définition de multiples réserves de carburant (fuel tanks).
- Les entrées DockRef, DockDir, DockRot ont été remplacées par Docklist, plus polyvalent, (voir ci-dessous), qui permet la configuration de multiples ports d'amarrage et de fréquences IDS.

```
BEGIN_DOCKLIST
   <Dock-spec 0>
   <Dock-spec 1>
   . . .
   <Dock-spec n-1>
END DOCKLIST
```

Où *<Dock-spec i>*:

<x_i> <y_i> <z_i> <dx_i> <dy_i> <dz_i> <rx_i> <ry_i> <rz_i> [<ids-channel>]

<x_i> <y_i> <z_i> est la position de référence du port d'amarrage dans les coordonnées locales du vaisseau.
<dx_i> <dy_i> <dz_i> est la direction dans laquelle un vaisseau approche le port d'amarrage dans la fenêtre de référence locale de la station.

<rx_i> <ry_i> <rz_i> est une direction de référence perpendiculaire à la direction d'approche utilisée pour aligner la rotation d'un vaisseau en approche le long de son axe longitudinal.

<ids-channel> est un paramètre optionnel qui permet de définir le canal pour un transmetteur IDS (Instrument Docking System) pour l'amarre. La valeur est un entier duquel la fréquence est calculée par :
$f = f_{min}$ + *<ids-channel>* * 0.05 kHz, où f_{min} = 108.0 kHz.
Le réglage de l'IDS peut être redéfini par des vaisseaux individuels via l'option IDS dans le fichier scénario.
Définir l'IDS dans le fichier de config est d'habitude seulement utilisé pour des objets avec une demande unique, par exemple des stations spatiales.

- La liste d'attachement est similaire à la docklist : elle permet de spécifier les points par lesquels les vaisseaux peuvent êtres connectés entre eux. Contrairement aux ports d'amarrage, les points d'attachements définissent les hiérarchies parent-enfant, et chaque point d'attachement est soit un port parent ou soit un port enfant. Pour plus de détails voir la section *Vessel attachment management* dans le manuel de référence API.

```
BEGIN_ATTACHMENT
   <Attach-spec 0>
   <Attach-spec 1>

   . . .
   <Attach-spec n-1>
END_ATTACHMENT
```

Où *<Attach-spec i>*:

<type> <x_i> <y_i> <z_i> <dx_i> <dy_i> <dz_i> <rx_i> <ry_i> <rz_i> <id>

<type> est un caractère simple : 'P' – "attache à un parent", ou 'C' – "attache à un enfant".

Les 9 prochaines entrées définissent la position d'attachement et la direction de la même manière que les ports d'amarrages.
<id> est une série de plus de 8 caractères utilisés pour définir la compatibilité entre des points d'attachements.
.

- **Inertia tensor *J*:** Etablit le moment angulaire et la vélocité angulaire: $\mathbf{L} = J \cdot \boldsymbol{\omega}$

$$J = \frac{1}{M}\int_{Vol} m(r)\begin{pmatrix} y(r)^2 + z(r)^2 & x(r)y(r) & x(r)z(r) \\ y(r)x(r) & x(r)^2 + z(r)^2 & y(r)z(r) \\ z(r)x(r) & z(r)y(r) & x(r)^2 + y(r)^2 \end{pmatrix} dr$$

Où M est la masse totale du vaisseau, l'intégration est sur le volume vaisseau. Notez que cette définition se normalise par M, donc l'unité de J est [m2]. Les principaux moments d'inerties (PMI) J_x, J_y, J_z que nécessite le fichier de configuration sont les éléments diagonaux de J dans une fenêtre de référence dans laquelle J est diagonal:

$$\hat{J} = \begin{pmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{pmatrix}$$

Le SDK contient un outil simple pour calculer le tenseur d'inertie pour un mesh donné: `Orbitersdk\utils\shipedit.exe`. L'outil nécessite des meshes se « comportant bien » (composés de surfaces compactes fermées) et assume une distribution de densité homogène dans le mesh. Le dernier n'est pas très réaliste, donc les résultats doivent êtres interprétés avec prudence.
Ils devraient servir de bon point de départ pour de l'expérimentation.

3.1 Fichiers de configuration pour des vaisseaux individuels

Un vaisseau nécessite seulement un fichier de définition individuel si ce n'est pas la demande d'une classe de vaisseau. Dans ce cas le format pour le fichier .cfg du vaisseau est identique aux fichiers .cfg de classe de vaisseaux décrit au-dessus.

4 Le fichier « mesh »

ORBITER utilise un format de fichier mesh propriétaire. Les fichiers mesh sont des fichiers textes ASCII. (Un format binaire va peut-être être introduit dans le future). Les fichiers mesh sont localisés dans le sous-répertoire `Meshes` à moins que l'entrée MeshDir dans Orbiter.cfg n'apparaisse dans un répertoire différent.

Les meshes d'ORBITER sont définis dans un système de coordonnées de gaucher. Les meshes devraient êtres orientés de manière à ce que le *nez du vaisseau* (ou plus précisément, *la direction de sa poussée principale*) pointe positivement dans la z-direction, l'axe x positif pointe à *droite*, et l'axe y pointe positivement vers le *haut*.

Les unités pour les coordonnées vertex sont en *mètres* [m].

Format du fichier mesh :

`MESHX1`	header
`GROUPS` <*n*>	<*n*>: number of groups

<group 1>	group spec 1
<group 2>	group spec 2
…	
<group n>	group spec *n*
MATERIALS *<m>*	*<m>:* number of materials
<mtrl-name 1>	material name 1
<mtrl-name 2>	material name 2
…	
<mtrl-name m>	material name *m*
<material 1>	material spec 1
<material 2>	material spec 2
…	
<material m>	material spec *m*
TEXTURES *<t>*	*<t>:* number of textures
<tex-name 1>	texture name 1
<tex-name 2>	texture name 2
…	
<tex-name t>	texture name *t*

Spécifications du groupe :

[LABEL *<label>*]	group label; optional
[MATERIAL *<i>*]	material index; optional
[TEXTURE *<j>*]	texture index; optional
[TEXWRAP *<wrap>*]	texture wrap mode: *<wrap>* = U or V or UV; optional
[NONORMAL]	“no normals” flag; see below; optional
[FLAG *<f>*]	multi-purpose bit-flags; see below; optional
GEOM *<nv>* *<nt>*	*<nv>:* vertex count, *<nt>:* triangle count
<vtx 0>	vertex spec 0
<vtx 1>	vertex spec 1
…	
<vtx nv-1>	vertex spec *nv*-1
<tri 0>	triangle spec 0
<tri 1>	triangle spec 1
…	
<tri nt-1>	triangle spec *nt*-1

Spécifications de Vertex :

<x> <y> <z> [<nx> <ny> <nz> [<tu> <tv>]]	
	<x> <y> <z>: vertex position
	<nx> <ny> <nz>: vertex normal (optional)
	<tu> <tv>: texture coordinates (optional)

Les «normals» manquants sont automatiquement calculés comme la moyenne des faces adjacentes de « normals ». Des coordonnées de texture sont nécessaires uniquement si le groupe utilise une texture.

Spécifications de Triangle :

<i> <j> <k>	vertex indices (zero-based). Left-hand face is rendered.

Spécifications de Material :

MATERIAL *<mtrl-name>*	material header
<dr> <dg> <db> <da>	Diffuse colour (RGBA)
<ar> <ag> <ab> <aa>	Ambient colour (RGBA)
<sr> <sg> <sb> <sa> <pow>	Specular colour (RGBA) and specular power (float)
<er> <eg> <eb> <ea>	Emissive colour (RGBA)

4.1 Les groupes « Mesh »

Les meshes sont divisés en groupes. Chaque groupe peut définir la spécification de son propre matériel et de sa texture. Par exemple, si vous voulez que différentes parties de l'objet aient des propriétés différentes de matériel, vous avez besoin de séparer, en conséquence, le mesh en groupes.

Chaque groupe contient :

- Une étiquette optionnelle (tag LABEL). L'étiquette doit-être un seul mot sans espaces. Il n'a pas d'effet direct sur le mesh, mais peut être utilisé pour associer un nom avec un group mesh. Les groupes nommés sont plus facile d'accès depuis l'intérieur du code d'un module de vaisseau que des index de groupe (ex. pour définir des animations etc.)
- Un index de matériel optionnel. «Indices >=1» sélectionne un matériel de la liste de matériel du mesh. « Index 0 » veut dire " matériel par défaut " (qui est blanc, diffus et opaque). Si le groupe ne spécifie pas d'index de matériel il hérite du matériel du groupe antérieur. Le premier groupe dans le mesh *doit* spécifier un index de matériel, sinon le résultat est non définis.
- Un index de texture optionnel. «Indices >=1» sélectionne une texture depuis la liste de texture du mesh. « Index 0 » signifie "pas de texture". Si le groupe ne spécifie pas d'index de texture il hérite de la texture du groupe antérieur. Le premier groupe dans le mesh *doit* spécifier un index de matériel, sinon le résultat est non défini.
- Un drapeau optionnel TEXWRAP. Cela définis comment les textures s'enveloppent autour de l'objet. "U" pousse les textures à s'envelopper dans la direction d'u-coordonnée dans l'espace texel, "V" enveloppe en direction de v-coordonnée, et "UV" enroule dans les deux directions. « Default » n'en engendre aucun.
- Un drapeau optionnel NONORMAL. Cela indique que les définitions vertex dans ce groupe ne contiennent pas de définitions normales, et les deux premiers nombres après le triplet de coordonnées vertex (x,y,z) sont interprétés comme des textures de paires de coordonnées (u,v).
- Une entrée optionnelle FLAG. Cela permet de spécifier un drapeau 32-bit défini par l'utilisateur (en format hex) dont l'interprétation dépend du contexte. Ci-dessous une liste de drapeaux actuellement reconnus par Orbiter:

Type de mesh	Drapeau	Interprétation
Vessel	0x00000001	Ne pas utiliser ce groupe pour afficher les ombres au sol
Vessel	0x00000002	Ne pas afficher ce groupe
Vessel	0x00000004	Ne pas appliquer d'éclairage avec ce groupe affiché
Vessel	0x00000008	Directive de mélange de texture : additif avec le fond

- Une spécification GEOM, définit le nombre de sommets et de triangles dans le groupe.
- Une liste vertex (voir ci-dessous)
- Une liste triangle (voir ci-dessous)

Les listes Vertex

Chaque groupe contient une liste vertex, définissant les positions, et optionnellement les directions du paramètre « normal » et les coordonnées de texture des sommets dans le groupe.
Chaque ligne dans la liste définit un vertex, et contient plus de 8 nombres à virgule flottante (séparés par des espaces)

- Les 3 premiers nombres contiennent les coordonnées cartésiennes du vertex (x,y,z) dans l'espace des coordonnées locales de l'objet. Les unités sont en mètres [m]
- Les 3 numéros suivants (si présents) contiennent la « vertex normal direction» (nx,ny,nz) (à moins que le groupe ait sélectionné le drapeau NONORMAL). La « normal direction » est la direction perpendiculaire à la surface du mesh à la position vertex. Orbiter a besoin de cela pour générer des jeux de lumières correctes. Si aucun « normals » ne sont spécifiés (ou si le drapeau NONORMAL est sélectionné) Orbiter devine la direction « normal » comme la moyenne des « normals » des triangles environnants. Cela fonctionne bien pour des surfaces lisses, mais devrait être évité pour des surfaces qui contiennent des bords tranchants. Les directions « normal » devraient êtres normalisées. Ex.: $\text{sqrt}(nx_2+ny_2+nz_2) = 1$.
- Les 2 numéros suivants (si présents) contiennent les coordonnées de texture vertex (u,v).

Les coordonnées de texture sont demandées seulement si le groupe utilise une texture (ex. comme texture index >=1).
Les coordonnées de texture définissent comment une texture rectangulaire 2D est disposée sur la surface de l'objet. Des coordonnées de texture (0,0) se réfèrent au coin gauche le plus bas de la texture.
(1,1) fait référence au coin droit le plus haut. Des coordonnées > 1 sont permises et engendrent des textures qui se répètent périodiquement.

Notes:

- Les sommets localisés sur des bords tranchants ou des coins nécessitent plusieurs entrées dans la liste vertex, parce qu'ils ont des directions « normal » multiples (en d'autres mots, les surfaces *ne sont pas différentiables* aux bords). Dans ce cas vous devriez toujours définir « normals » dans le fichier mesh, et ne pas laisser Orbiter les générer pour vous. Sinon les bords vont apparaître lisses de manières irréalistes.
- De même, des sommets avec des coordonnées vertex multiples (ex. dans le coin entre deux toiles de textures) ont besoin d'entrées multiples dans la liste de vertex.

Les listes Triangle

La liste du groupe triangle suit immédiatement sous la liste vertex. Elle définit les triangles qui composent la surface du groupe du mesh.

- Chaque ligne dans la liste définit un triangle et relate 3 numéros entiers (i,j,k). Chacun des ces nombres spécifient un vertex de la liste du group vertex (en commençant par 0).
- Seulement le coté dans le « sens d'une montre » (CW (Clock Wise)) de chaque triangle est affiché : le coté qui, si vous le regardez, a les sommets arrangés dans le sens des aiguilles d'une montre. Le coté opposé « anti horaire » (CCW) est invisible.
- Si vous avez besoin d'afficher chaque coté d'un triangle (ex. pour une fine plaque) vous devez définir deux triangles.
- Si vous voulez retourner le coté affiché d'un triangle (ex. pour corriger les artefacts à l'envers) vous devez réarranger les indices du triangle de cette manière : (i,j,k) -> (i,k,j)

4.2 Liste du matériel

« Materials » permet de spécifier les propriétés de l'homogénéité de l'éclairage d'un groupe mesh.
La liste de matériel est constituée de :

- Une ligne de tête, `MATERIALS <m>`, qui définit le nombre <m> de matériaux.
- Une liste de *noms* de matériaux (material *names)*.
- Une liste de *propriétés* des matériaux (material *properties)*.

Chaque spécification de propriété de matériel consiste en 4 quadruplets RGBA, où R, G et B définissent les composants rouge, vert et bleu. A est l'opacité. Les valeurs RGB devraient êtres comprises entre 0 et 1, mais peuvent êtres > 1 pour des effets spéciaux. La valeur de A doit être comprise entre 0 (totalement transparent) et 1 (totalement opaque).

- La première ligne spécifie la *couleur diffuse* du *matériel*. C'est la couleur qui est diffusée (dans toutes les directions) reflétée par une surface illuminée.
- La seconde ligne spécifie la *couleur ambiante du matériel*. C'est la couleur d'une surface non éclairée.
- La troisième ligne spécifie la couleur spéculaire. C'est la couleur de la lumière reflétée par une surface polie dans un faisceau étroit. L'entrée *power* spécifie la largeur du cône dans lequel la lumière spéculaire est reflétée. Des valeurs importantes signifient un cône plus étroit, des réflexions plus tranchantes.
  Les valeurs typiques sont autour de 10. Si elles sont omises, La valeur de « power » par défaut est de 0.
- La quatrième ligne spécifie la *couleur émissive*. C'est la couleur de la lumière émise par une surface brillante.

4.3 Liste de texture

La liste de texture contient les noms des fichiers texture utilisés par les divers groupes mesh.
Les noms de texture devraient contenir les extensions de fichiers ".dds" mais pas les chemins des répertoires.
Les textures doivent êtres localisés dans le sous répertoire `Textures` d'Orbiter.

Notes:

- Les textures doivent êtres en format DDS ("Direct Draw Surface"). L'outil DirectX SDK, `dxtex`, qui est inclus dans le package d'Orbiter SDK, permet de convertir des bitmaps BMP en DDS.
- Vous devriez enregistrer les textures soit en format compressé DXT1 (textures opaques ou des textures avec des transparences binaires), ou en format compressé DXT5 (pour les textures avec une transparence continuelle).
- Pour un maximum de compatibilité, évitez des textures plus larges que 256x256 pixels, à cause des limitations de certaines vielles cartes graphiques.
- Si une texture est mise à jour trop dynamiquement durant la simulation (ex. des panneaux d'instruments dans les cockpits virtuels), le nom de texture devrait être suivi par le drapeau 'D'.
  Orbiter va décompresser ces textures pour permettre des dynamiques de mises à jour plus efficientes.

4.4 Optimisation de performance

Pour réaliser les meilleurs résultats avec votre nouveau mesh, considérez les points suivants:

- Les groupes de texture qui utilisent la même texture devraient êtres enregistrés en séquence dans le mesh. Des changements non nécessaires entre les textures peuvent diminuer la performance si les textures doivent êtres transférées dedans et dehors de la mémoire vidéo.
- Dans une séquence de groupes utilisant les mêmes textures, les groupes qui utilisent le même matériel devraient êtres stockées en séquence. Là encore, cela évite d'avoir besoin de changer les paramètres d'affichages.
- Evitez les grands nombres de très petits groupes. Si des petits groupes utilisent les mêmes paramètres (matériel, texture, etc.) Ils devraient êtres rassemblés dans un seul groupe.
- Les groupes qui utilisent des matériaux ou des textures transparentes devraient êtres enregistrés en fin de mesh. Si des groupes transparents se chevauchent, les plus à l'intérieur devraient être listés avant ceux le plus à l'extérieur.
  Pour afficher la transparence correctement, DirectX à besoin de la scène vue à travers l'objet transparent pour être complètement construit avant que l'objet transparent soit lui-même affiché. Tout objet affiché après l'objet transparent va être masqué par lui.
- Les objets avec de la transparence et de la réflexion spéculaire sont plus couteux à afficher que les objets opaques et diffus, donc utilisez ces caractères économiquement.
- Et le plus important, *garder le compte vertex bas!* (Voir section 6.1)

4.5 Convertisseurs de Mesh

Si vous voulez convertir un model existant dans un mesh d'Orbiter, vérifiez auprès du forum web d'Orbiter s'il existe des convertisseurs de mesh créés par d'autres utilisateurs. Il y a actuellement un convertisseur qui convertit depuis un format Truespace asc, que beaucoup d'éditeurs 3D peuvent exporter. Si vous avez écrit votre propre éditeur ou convertisseur mesh, publiez-le !

4.6 Utilitaires de mesh

Le SDK d'Orbiter contient quelques utilitaires qui aident à extraire des données depuis des fichiers mesh.
Ils sont situés dans le dossier Orbitersdk\utils.

shipedit : extrait des informations géométriques d'un mesh qui sont utilisés pour définir les paramètres physiques pour des modules de vaisseau. Ils incluent l'étendue de la boîte englobante, le volume, les sections transversales, et le tenseur d'inertie pour une distribution homogène de la densité.
meshc : compilateur de mesh. Eventuellement cela peut être étendu à la conversion de fichiers mesh en texte vers un format binaire (Pour un stockage plus compacte et chargement plus rapide) mais actuellement il extrait seulement des paramètres mesh dans un fichier de tête C qui peut être inclus dans un projet de module de vaisseau pour un accès pratique à des groupes de mesh nommés.

5 Créer un scénario

Pour actuellement faire voler votre nouvelle création dans Orbiter, vous avez besoin de créer un scénario qui contient un (ou plusieurs) vaisseau(x) de nouvelle classe. La manière la plus simple de faire cela est d'éditer un des fichiers de scénario existant dans les sous répertoires de « Scenarios ». Par exemple, pour essayer une nouvelle classe de vaisseau, vous pouvez :

- Ouvrir le fichier "Habana spaceport.scn" dans notepad.
- Remplacer la ligne
  `GL-01:DeltaGlider` par
  `GL-01:<new class>`
  Où <new class> est le nom de votre nouvelle classe de vaisseau.
- Sauvez le scénario sous un nouveau nom.

Lorsque vous lancez ce scénario modifié, vous allez vous trouver dans le cockpit de votre nouveau vaisseau, parqué sur un pad de lancement du Habana International Spaceport.

6 Publier des addons

Maintenant que vous avez crée et testé votre nouveau vaisseau, Vous souhaitez le partager avec le reste de la communauté d'Orbiter. Voici comment le faire :

6.1 La fonction est plus importante que le style

Il peut être tentant de créer un modèle 3D extrêmement détaillé avec des dizaines de milliers de sommets de mesh, et mégabits de textures – mais ne le faites pas ! Rappelez-vous que beaucoup de personnes ont peut-être des ordinateurs moins performants que vous, et que le plus sexy des vaisseaux est sans valeur s'il dégrade la fluidité de l'affichage jusqu'à rendre Orbiter injouable. Aussi, votre modèle peut concourir avec des dizaines ou des centaines d'autres objets pour économiser les cycles du processeur.
Donc, ayez un *compte de polygone bas*. Créer des objets agréables à regarder avec des meshes de basse-résolutions est le sommet de l'art de la modélisation de 3D pour des applications en temps réel. Si vous importez un modèle existant dans Orbiter, voyez si votre éditeur 3D a une option pour réduire le compte de polygone (quelque chose appelé *optimisation* or *decimation*) avant de convertir en format mesh d'Orbiter. Comme ligne de conduite, je voudrais suggérer de garder le compte vertex en dessous de 10 000 sommets pour chaque vaisseau spatial.

De même, essayez de limiter les textures utilisées par votre objet. Les cartes graphiques ont une mémoire de texture limitée, et votre vaisseau va partager cet espace avec beaucoup d'autres objets.
Donc je suggère :

- Un petit nombre de « maps » de texture, aux dimensions 256x256 ou plus petit.
- L'utilisation de « maps » de textures génériques (ex. des textures pour des panneaux solaires etc.) qui peuvent être réutilisées par d'autres modèles, aident à réduire les besoins en mémoire de texture. Regardez le répertoire de textures d'Orbiter, pour voir si une texture standard existante d'Orbiter est utilisable pour votre modèle.
- Rappelez-vous que quelques vieilles cartes graphiques ne supportent pas de textures plus larges que 256x256. Evitez tout ce qui est plus large si vous voulez assurer la compatibilité.

Ces limites sont de simples suggestions pour publier des addons qui assurent une performance raisonnable sur de vieux ordinateurs – vous êtes libre de les ignorer. Vous devriez toujours mentionner la complexité de votre modèle (ex. le compte vertex) et les possibles incompatibilités dans le fichier « readme » qui accompagne l'addon, pour aider les utilisateurs à décider s'ils peuvent utiliser votre addon.
Bien sur il n'y a pas de limitations pour des modèles créés pour votre usage privé.

6.2 Créer un « package d'addon »

Créer un fichier zip qui contient tous les composants de votre nouveau modèle (fichier readme, fichier configuration, mesh, textures, scénario). Le fichier zip devrait contenir le répertoire d'information des fichiers, de manière à ce que tous finissent en bonne place lorsque l'utilisateur décompresse l'archive dans le répertoire de base d'Orbiter.
Ne PAS inclure des versions de fichiers standards d'Orbiter modifiés (comme des fichiers de configuration de planète ou des fichiers de configuration de système solaire) qui peuvent réécrire des fichiers déjà existants. Si votre package a besoin de modifier des fichiers standards cela devrait être décrit dans le fichier « readme ». Testez que le package se décompresse et fonctionne bien avant de le publier (idéalement dans une nouvelle installation d'Orbiter)

Soyez sûr que votre package contient un fichier « readme » avec au moins les informations suivantes:

- Votre nom et email
- La version d'Orbiter pour laquelle le package a été écrite.
- Une description du package (quel genre de vaisseau, etc.)
- Une liste des fichiers dans le package.
- Les instructions d'installation, incluant les changements demandés aux fichiers de configuration standards d'Orbiter.
- Un (approximatif) compte vertex de manière à ce que les utilisateurs puissent estimer les conséquences sur les performances.

6.3 Le mettre sur le web

Mettez votre nouveau package sur un site web (avec une courte description et une capture d'écran optionnelle) et parlez-en autour de vous ! Si vous n'avez pas d'espace web, proposez-le à un des utilisateurs d'Orbiter ayant un espace prévu pour le déposer.
La page des "Sites associés" sur le site de base d'Orbiter contient une liste de dépositoires d'addons pour Orbiter. Une des archives d'Orbiter la plus populaire peut être trouvée sur AVSIM (www.avsim.com). Pour une plus large publicité, vous devriez mettre une note avec lien dans le forum d'addons d'Orbiter, ou envoyer un e-mail à la liste de diffusion d'Orbiter. Finalement, vous voulez peut-être proposer qu'une capture d'écran de votre addon soit publiée sur le site web officiel d'Orbiter. Voyez la page de la galerie d'Orbiter pour un lien vers les directives de demandes.

Note : Traduction faite le 10.07.2022 de l'original en Anglais vers le Français par SpaceTravel. (thierry.sbrugnera@protonmail.com)