Représentations visuelles de la **structure d'un système stellaire**

ÉPREUVE INTEGRÉE . BACHELIER EN INFORMATIQUE DE GESTION

Christophe Jossart
EPFC | Année académique 2010-2011

1_Objectifs et moyens [p.3]

Concept . Techniques . Méthodologie . Analyse

2_Mise en oeuvre [p.14]

Plan de l'application . Découpage en couches . Formalisme

3_Détails de l'implémentation [p.21]

Tests . Structure commune d'une application . Gestion de projet . Détail de chaque module

4_Observations [p.38]

Informatique et graphisme . ActionScript 3 . API 3D . Performances . Animation . Design Pattern . Algorithmes courrament rencontrés en graphisme

5_Réflexions [p.61]

Formats propriétaires . Pourquoi AS3 . Interopérabilité

6_Sources [p.65]

Bibliographie . Inspiration . Crédits

Annexes

Diagrammes UML

Fichiers XML de configuration (système stellaire, ...)

Sélection d'algorithmes

Téléchargement du code de l'application et site de démo

1_Objectifs et moyens

1.CONCEPT

Le concept générique est de présenter les relations que peuvent entretenir l'informatique et le graphisme au travers d'une thématique : l'astronomie.
Disposant d'une formation dans ces trois domaines, ce travail est donc l'opportunité de les réunir au travers d'une intégration.

Le fil conducteur de cette intégration est définit par la représentation d'un système stellaire et le traitement des informations résultant des interactions entre ses différents objets, via des solutions open source.

L'enjeu étant non pas la production mais la recherche, cette thématique est donc un prétexte à l'étude des multiples outils (API, Framework, ...) mis à notre disposition pour illustrer visuellement et interactivement un propos, via une interface graphique riche. Cela nous amène à considérer, par exemple, les algorithmes ayant trait aux applications graphiques.

La mise en oeuvre permettra donc d'aborder une série de techniques et concepts liés à un système d'information (design pattern, manipulation de structures de données, algorithmique, performances, programmation réseau, ...).

Ma volonté est de laisser les portes ouvertes à de nouvelles directions, les bases de l'intégration ont donc été fondées en ce sens via :

1. Un modèle robuste de la représentation de la matière (cf. Annexes : diagramme UML).
2. Un framework qui permettra d'afficher aisément tout type d'applicatif dans une interface graphique cohérente.

Le projet est donc assez ouvert pour permettre de multiples extensions ultérieures, il a cependant été nécessaire de limiter les domaines d'investigation pour cette première phase de travail, afin de pouvoir illustrer cette démarche au travers d'une implémentation.

Afin de trouver les sujets de l'intégration, je me suis posé quelques questions quant à l'application de concepts informatiques au thème principal :

Par exemple :

- Le design pattern composite pourrait-il convenir comme modèle de représentation de l'univers ?
- Un algorithme de calcul des plus courts chemins (Dijkstra, ...) serait-il appliquable pour trouver un moyen d'optimiser les catapultes gravitationnelles, ... ?
- Comment traiter les données multimédia (vidéos, photos, liens, flux rss, réseaux sociaux, ...) de manière automatisée afin de dynamiser une application ?
- Quels seraient les moyens pour explorer des concepts tels que les systèmes de particules et la physique appliquée en général ?

Outre ces questions précises j'ai également été stimulé par une préoccupation en tant que développeur ActionScript professionnel : ce langage (et l'utilisation de la Flash Platform en général) étant au centre de la polémique depuis la sortie d'HTML 5, je trouve intéressant de pousser celui-ci dans ses limites afin de constater s'il peut encore offrir un résultat (aujourd'hui) non atteignable dans l'environnement web par une autre technologie.

En ce sens et afin de répondre aux questions ci-dessus, j'ai donc exploré la richesse de l'écosystème ActionScript 3 (Langage, Framework, API, interopérabilité, ...).

Par voie de conséquence, cela nous amène à considérer les solutions open source dans un cadre intéressant (ce langage étant un format propriétaire !) tout en ayant une solution alternative émergeante (HTML 5).

1.1. Fonctionnalités principales

L'objectif initial, qui est présenté à la fin de cette rubrique, était centré sur les réactions de fusion dans une étoile.
Ce thème a déjà été traité par l'Observatoire de Paris-Meudon via une bibliothèque Java sur mesure (Simulab).

Il était inutile de simplement recommencer un travail existant, aussi, je me suis concentré sur la conception d'un système stellaire aisément modifiable, me focalisant non plus sur les étoiles, mais plutôt sur les sous-produits externes en résultant (objets, éléments chimiques, ...).

Je laisse donc les premières réflexions ci-dessous uniquement pour archive, celles-ci pourront éventuellement servir à poursuivre un travail personnel. Une piste est en cours grâce à une solution de traduction automatisée Java / ActionScript 3 (j2as). J'ai cependant rejoint cette première idée d'une certaine manière en utilisant l'algorithme de Perlin afin d'obtenir une représentation visuelle, animée, du soleil (voir point 3.5. Détails de l'implémentation).

L'essentiel de mon travail a donc été divisé en plusieurs tâches:
1. Réaliser le framework de présentation des divers applicatifs
2. Modéliser (en UML) la matière
3. Développer les modules de représentations visuelles
4. Etudier les solutions open source permettant de m'aider dans ces tâches
5. Intégrer le tout en un ensemble cohérent en utilisant cette l'application principale comme une aplication cliente du framework de présentation

Devant l'ampleur de la tâche, afin de ne pas perdre le fil rouge, j'ai également conçu un site internet faisant office de gestionnaire de projet et d'outil de rédaction du rapport (http://ei.hypercube.be).

Pour archive : première réflexion concernant les réactions de fusion d'une étoile :

Données :
Evolution chimique et physique dans le temps d'une étoile d'une masse solaire et extrapolation des résultats pour les masses inférieures et supérieures (inférieures : naines / supérieures : supernovae).

- Masse et volume
- Concentration des éléments
- Luminosité
- ...

Contrôles utilisateur :

- Modulation de la masse initiale (qui est d'une masse solaire)
- Accélération et décélération du temps (déplacement dans le temps)
- Changement de vue
- ...

Représentations :
Possibilité de modifier les vues sur les données :

- Evolution de la masse et du volume via une sphère (3D) mise en situation dans le système solaire afin de disposer d'une comparaison d'échelle
- Différentes étapes de fusion (cycles de la nucléosynthèse stellaire : fusion de l'hydrogène, fusion de l'hélium, CNO, ...) via une représentation des atomes
- Evolution sur le diagramme de Hertzsprung-Russell (luminosité/température)
- Composition chimique (sur le tableau périodique)
- Situation dans le système solaire
- ...

1.2. Autres extensions possibles

- Etude des API C++ de physique telles que Box2D, portées sur la machine virtuelle Flash grâce à Alchemy (cf. 5.3. Interopérabilité)
- Etude des systèmes de particules tels que Flint
- Manipulation des contrôles avec d'autres entrées que le clavier : données de la webcam (mouvement), du microphone (voix).
- Représentation de la matière microscopique, par extension à la matière macroscopique (cf 1.4. Analyse) : utilisation du format standard .mol comme source de données à la modélisation 3D
- Pour l'application de représentation d'un système stellaire : éditeur de système (CMS), mise en réseau, exploitation d'un detailLevel (cf. fichier de configuration XML en annexe) pour la repésentation d'un galaxie, d'une superstructure de l'univers
- ...

2.TECHNIQUES MISES EN OEUVRE

2.1. Langages, formats, framework et API

→ ActionScript 3

Outre ce langage orienté objet, une série d'API écrites en AS3 sont au centre de l'intégration, elles sont toutes détaillées dans la suite de ce rapport.

- **Papervision3D :** Historiquement, première API 3D accessible sous la machine virtuelle ActionScript (AVM 2)
- **Away3D :** API 3D qui a succédé à Papervision3D
- **Tweener :** API d'interpolation de mouvement utilisant les équations d'accélération
- **Google Maps pour Flash :** API de cartographie (actuellement en développement) de Mars, de la Lune et du Deep Space Field de Hubble
- **Polygonal :** types de données abstraits

→ PHP 5.2 / MySQL 5

Ce couple « indissociable » a été utilisé pour le gestionnaire de projet (cf. 3.4. Gestion et présentation du projet).
Vu que ces outils et la gestion du projet n'est pas au centre des préoccupations, afin de monter en abstraction, j'ai utilisé le CMS/ Framework **Drupal**.

AMFPHP / Zend AMF : RPC permettant une sérialisation des types de données ActionScript vers le côté serveur et établit alors une communication optimisée avec les objets des classes PHP (dans notre cas). Ceci permet un gain de performance et de temps de développement par rapport à des formats « passerelle » de type XML ou JSON.

→ JavaScript, HTML, CSS

Ceux-ci n'ont joué qu'un rôle mineur : l'intégration de la machine virtuelle au navigateur web. L'API **SWFObject** a été utilisée.
Ce rôle ne doit cependant pas être négligé vu que cette intégration permet :

1. D'assurer une prise en charge de la machine virtuelle sous tous les navigateurs (Internet Explorer, Firefox, Safari, Opera, ...) et tous les OS (Windows / Unix).
2. De mettre à jour le Flash Player si la version requise (10+) n'est pas installée sur la machine cliente
3. De permettre un affichage plein écran

→ XML

Ce format a permit de concrétiser le modèle de l'application.
Les tags XML définissent les classes ActionScript à instancier, leurs relations et les paramètres éventuels à leur fournir.

XML a été utilisé principalement dans 3 cas :

1. Structure de l'application principale (menus et écrans associés).
2. Définition de la structure d'un système stellaire (objets, textures, coefficients de correction, ...)
3. Spécialisation de XML par le format **RSS**

2.2. Machine virtuelle ActionScript (AVM)

Il existe deux possibilités d'exploiter l'AVM 2
(seule la version 2 prend en charge ActionScript 3) :

1. L'utiliser en tant que plugin de navigateur via **Flash Player**, à la manière d'un applet java.

2. L'utiliser en temps que runtime desktop via **Adobe Integrated Runtime** (AIR). Tout comme Java, celui-ci assure une compatibilité cross-platform (Windows / Unix), dispose d'une API permettant de gérer des révisions de l'application, ...
   Le fonctionnement est également le même que la JVM (utilisation du bytecode) et compilation à la volée (JIT).

2.3. Choix des IDE et outils

→ Flash Platform et ActionScript

Relations et rôles respectifs de Flash Builder (aka Flex 4) et de Flash Professionnal CS5

Flash Builder est un outil étendant Eclipse et est dédié exclusivement au développement.

Flash Professional permet le développement et confirme ce rôle depuis sa version 5 (prise en charge de l'autocomplétion), mais ne réalise aucune vérification structurelle.

Par contre, ce dernier dispose d'un vaste panel d'outils de création graphique :

- Dessin vectoriel et bitmap
- Editeur d'animation
- Prise en charge (très) basique de la 3D
- Prise en charge du format Photoshop (structure en calques).

Depuis Flash Builder (version 4 de Flex), il est possible de lier les processus d'édition aux processus de création (Flash).

Vu que Flash permet de créer, directement dans l'environnement auteur graphique, des classes (qui sont la plupart du temps des sous-classes de Movie Clip), une fonctionnalité de vérification des types existe entre les deux outils. Ceci constitue une excellente aide à l'édition, toutefois cette fonctionnalité ralentit considérable-

ment la compilation vu que Flash Builder vérifie toutes les classes créés dans l'environnement graphique Flash Professional !

Auparavant (Flex 3), il fallait déclarer chaque classe graphique comme étant une classe plus générale (p.ex. MovieClip) pour éviter une erreur à la compilation.

Notons toutefois que cela ne concerne que les classes « graphiques » étendant les classes natives sans être définies selon une classe créée par l'auteur.

P.ex. un élément purement décoratif ne se verra associer aucun comportement particulier et étendra donc la classe Sprite (pas de nécessité de ligne du temps) et une animation type dessin animé pur étendra la classe MovieClip.

Alternatives publiées par Adobe

Dans la même collection d'outils Adobe (Creative Suite), il existe une solution destinée à simplifier le workflow et les allers-retours entre développeurs et designers d'application : Flash Catalyst.

Cette solution est exclusivement destinée aux designers et se veut la prolongation d'un outil tel que Photoshop, ne perdant pas le designer avec des outils d'édition de code.

Je pense toutefois que cela jette la plus grande confusion, vu qu'il existait préalablement un outil complémentaire à Flex 3 remplissant ce rôle : Fireworks. Pour ajouter à la confusion, ce dernier outil fait en quelque sorte double emploi avec Photoshop...

On impute cette situation à la fusion de Macromedia et Adobe :

Macromedia disposait de Flash et Fireworks
Adobe disposait de Photoshop.

Un bel exemple de «compatibilité ascendante» : le choix d'Adobe a été de maintenir tous les outils afin de ne pas perdre d'utilisateurs, toutefois, pour un nouveau venu dans l'édition web, le choix de la bonne solution s'avère peu évident !

Alternatives publiées par d'autres éditeurs

Il existe également d'autres IDE gratuites (Flash Develop) ou payantes (FDT).

J'ai choisi celle de l'éditeur du langage pour un souci de facilité d'échange avec l'environnement Flash, ainsi que les autres outils de Adobe Creative Suite (CS5) : Photoshop, Illustrator, InDesign... Ce choix a uniquement été motivé par le fait que l'application allait être construite avec un haut niveau de personnalisation graphique.

→ Utilitaires ActionScript

Tests unitaires : FlexUnit

Pour la création 3D : prefab3D, Visual PV3D

Pour la publication AIR : LaunchPad

Traitement Bitmap : Pixel Bender Toolkit (API et IDE entièrement dédié aux filtres Bitmap et autres transformations réalisées sur les pixels) , ...

→ PHP, SQL, JavaScript, HTML

Dreamweaver CS5, NetBeans

→ Modélisation UML

Argo, Visual Paradigm

→ Gestionnaire de versions

GIT (GitHub)

→ Mise en page du rapport

InDesign CS5
Cet outil a été retenu car il permet notamment d'exporter dans les formats XML, ePub (cf. 4.1. Informatique et graphisme) et PostScript (dont PDF).

3. MÉTHODOLOGIE

J'ai appliqué la même méthodologie que celle qui s'est découverte être la plus efficace dans mon milieu professionnel.
Celle-ci emprunte des idées des méthodolgies RAD / Agile, XP, RUP, ...

1. Définition des concepts métiers, qui vont piloter la réflexion du modèle de l'application
2. Wireframe des vues des différents écrans afin de ne pas trop se perdre dans l'abstraction
3. Prototypage rapide du résultat à atteindre afin d'obtenir les séquences des actions, écrans, ...
4. Découpage MVC afin d'établir un diagramme de classes
5. Implémentation module par module
6. Tests itératifs
7. Implémentation de la totalité
8. Tests finaux
9. Mise en production
10. Maintenance
11. Révisions

Dans ce cas, j'étais mon propre client. Dans la pratique, nous impliquons le client dans chacune des phases en utilisant des formalismes tels que UML, des maquettes d'écran (avec une définition allant du wireframe au montage graphique complet, avant toute intégration).

4. ANALYSE

Outre l'analyse textuelle introduite préalablement et également précisée dans chaque module, l'objectif de cette section est de prendre un peu de hauteur en présentant trois niveaux de détails, via des diagrammes de classe UML. Ceux-ci sont disponibles en annexe.

Niveau de détail 1 :

L'application de gestion des écrans.
Classes responsables de la présentation d'un écran de l'application (et donc la structure nécessaire à l'affichage et à la circulation dans le site). Nous prenons également l'exemple du premier écran.

Modules concernés et détails de l'analyse :
Gestionnaire de layout, Menu principal d'une application

Niveau de détail 2 :

Présentation d'une application qu'il est possible de charger comme composant d'un écran: voyage dans le système solaire.

Module concerné et détails de l'analyse :
Voyage manuel dans le système solaire

Niveau de détail 3 :

Présentation du Model utilisé en partie pour le niveau de détail précédent.

Modules concernés et détails de l'analyse :
Voyage manuel dans le système solaire,
Réactions de fusion d'une étoile

Ces trois diagrammes niveaux ne représentent pas l'ensemble du système, mais permettent d'aisément d'extrapoler aux autres modules.

2_Mise en oeuvre

1. PLAN DE L'APPLICATION

Note préalable :
Vu le temps engagé
1. Dans la recherche, qui reste l'enjeu principal (étude des API 3D, mais également collecte des données astronomiques telles que la cartographie Google Maps, les textures des planètes, ...),
2. Dans la conception d'un framework de présentation sur mesure

j'ai préféré privilégier la qualité au volume de la production.
Une intégration partielle, mais néamoins avancée, de ce qui est présenté ci-dessous a été réalisé.
Je présente donc ici mon intention initiale, en mentionnant ce qu'il reste à intégrer dans une version ultérieure de l'application.

L'allégorie est celle du voyage, les rubriques sont faites pour être consultées selon une séquence, mais il est également possible de ne pas la respecter.

Plan général

1. L'actualité, afin de plonger dans le vif du sujet.
2. Présentation globale du système solaire
3. Comment en est-on arrivé là ? Présentation d'une étoile.
4. Une petite mise au point via la cartographie
5. Après ces quelques éléments, petit examen (quiz) avant le grand départ...

Plan détaillé

1. Rubrique multimédia d'actualités puisées sur l'internet

Il s'agit du point d'entrée de l'application.
Afin de ne pas rendre cet écran figé, une variation automatisée des contenus est prévue via différents moyens :

→ **Texte de présentation et dernières mises à jour**
Synchronisation avec le CMS Drupal

→ **Agrégation de flux RSS provenant de l'ESA**

→ **Tweets en rapport avec la thématique** [A réaliser]

→ **Succession de visuels affichés aléatoirement en fond d'écran**

→ **Sous-menu lié à cette rubrique**

1. **Premier contact :** lien vers le premier écran
2. **Plan de vol :** plan 3D de l'application, interactif [A réaliser]
3. **Journal de bord :** galerie photo 3D liée à une thématique avec vidéo plein écran en arrière plan.
   Mise en oeuvre par l'API de Flickr [A réaliser]
4. **Infosphère :** bookmarks partagés (Liaison à l'API de Delicious). [A réaliser]

2. Rubrique « Voyager / Sonder le système solaire »

→ **Diverses présentations du système solaire.**

1. **Perspective : une première approche**
   Différents points de vue (caméras) des planètes alignées sur un même axe, afin de percevoir immédiatement les échelles de grandeur et de distance. Utilisation de l'**API Away3D**.
2. **Notre système : fiches techniques :**
   Présentation de chaque planète du système solaire, avec leurs données techniques.
   [A finaliser - Composant de panneaux coulissants réalisé].
   Plus haut niveau de détail que le module «Perpective» : chaque objet est présenté avec ses satellites, ...
   Principe de navigation latéral, par panneaux à scroller, highlight dans le footer de la situation dans le système solaire. Illustration du **design pattern composite** (système = étoile + satellites disposant eux-mêmes de satellites / faire la relation avec les atomes).
   Export de dossiers pédagogiques (**API AlivePDF**).

→ **Mise en situation : de l'infiniment grand à l'infiniment petit :** Représentation de la superstructure de l'univers en passant par la galaxie, le système stellaire, planétaire, ...jusqu'à l'atome. [A réaliser]

3. Rubrique « Le Laboratoire / Fusion et création »

La réflexion initiale est laisée ci-dessous pour archive et développement ultérieur (cf. 1.1. Fonctionnalités principales).

Celle-ci a été remplacée par une illustration de l'utilisation de l'algorithme du bruit de Perlin et une description textuelle présentée dans un composant de navigation par onglets, réalisé sur mesure (équivalent du Tabbed Pane de Java Swing).

Réflexion initiale :

Présentation détaillée de la vie des étoiles.
Le Soleil est pris comme étalon, par défaut, mais il est possible de moduler la masse stellaire initiale afin de constater les différentes réactions de fusion possibles.

Plusieurs vues, qui sont toutes Observer des réactions de fusion, c'est-à-dire le modèle (la modification des données s'opérant dans le temps).

1. Rotation 3D + Pixel Bender
   Away3D + AnimatedBitmap
2. Dimension dans le système
3. Diagramme HR (luminosité, température, type spectral, ...)
4. Histogramme de la concentration des éléments
   (via Google Motion Chart)
5. Tableau périodique circulaire, représentant les éléments générés par les réactions de fusion

Les données sont calculées par une application Java existante.
L'échange des données se réalise via la classe ExternalInterface.

Il serait possible de prendre une «capture d'écran», via l'export à la volée des données Bitmap au format jpg.

Modèle : réactions de fusion

On distinguera les étapes de fusion possible pour tous les types d'étoiles puis on définira les différents embranchements possibles que ce soit pour la nucléosynthèse comme (par voie de conséquence) pour le type d'objet qui en résultera (naine blanche, trou noir, ...)

LA NUCLEOSYNTHESE STELLAIRE

1. La chaîne proton-proton (T → 10 000 000 K)
   Fusion de H en He (4H1 → He4)
   Fusion de deux noyaux de H ensemble (= 2 protons)
   H1 + H1 = He2 (deutérium)
   ...
2. Cycle CNO (T → 15 000 000 K)
3. Processus triple alpha (T → 100 000 000 K)
4. Fusion carbone-carbone (T → 600 000 000 K)
5. Fusion Oxygène - Oxygène (T → 1 500 000 000 K)
6. Fusion du silicium (T→ 3 000 000 000 K)
7. Formation des autres éléments

ETAPES DE LA VIE D'UNE ETOILE

1. Nuage moléculaire
2. Proto-étoile
3. Etoile figurant dans la séquence principale du diagramme HR
4. Géante rouge
5. Différentes possibilités de fin : Naine blanche, trou noir, ...

4. Rubrique «L'Observatoire / Cartographie du Ciel»

Présentation de différentes cartes astronomiques en cours d'élaboration par Google, sur base de l'API Google Map / Google Earth

1. Hubble deep space field
2. La Terre
3. La Lune
4. Mars

5. Rubrique «La Passerelle / Le début du voyage»

→ « Starfleet »

Quiz représentant un pseudo examen permettant de s'enrôler dans Starfleet (petit clin d'oeil à la série Star Trek).
Le résultat de cet « examen » permettra de déterminer si l'écran suivant sera un voyage manuel ou automatique.
[A réaliser]

→ « En avant toute ! »

Voyage avec le vaisseau Enterprise (Star Trek, deuxième clin d'oeil), dans le système solaire.
Visite de chaque planète :
Un mode manuel et un mode automatique pourront donc être choisis.
S'il s'agit du mode manuel, calcul du voyage avec un algorithme permettant de tenir compte des catapultes gravitationnelles.

Affichage des données telles que : vitesse, km/AL parcourus, ...
Une extension possible serait la visite en réseau (multi-utilisateur). L'utilisation de Flash Media Server servirait de support à cette mise en oeuvre.

Utilisation de SpringCamera
(si mode manuel → API Papervision3D)

Utilisation de PathAnimator
(si pilote automatique → API Away3D).

Exploitation de Google Sketchup pour la modélisation de l'Enterprise.

[A finaliser : voyage automatique et intégration des modèles Google Sketchup]

2. DÉCOUPAGE EN COUCHES

Je souhaite souligner ici un des moyens (évident) du paradigme orienté objet : la réutilisation.

Du point de vue de l'approche, l'idée est également de démontrer qu'on peut très bien ne pas tomber dans la stagnation (idée de recyclage, «d'utilisateur d'API» / «assembleur de legos») mais plutôt apporter une continuité au travail des autres.
Ce débat ne date bien sûr pas de la POO...
Ceci dit, il est également remis à l'actualité par des outils de création de Mashup tels que Yahoo Pipe, ...

L'application est donc basée sur une « pile de classes » :

1. **Le noyau du langage ActionScript 3**
   P.ex.: La gestion de la DisplayList 2D, la classe Math, les gestionnaires d'événements natifs, ...
2. **Les Framework API externes**
   P.ex.: La gestion de la DisplayList 3D, ...
3. **Les classes utilitaires créées par des tiers**
   P.ex.: Structures de données telles que les Listes, gestionnaire de layout fluide, ...
4. **Les classes utilitaires personnelles**
   P.ex.: Gestionnaires d'écrans et d'application, outils de test de performance, de debug, ...
5. **Les classes dédiées uniquement à cette application**
   P.ex.: Les classes de représentation d'un étoile, ...

3. FORMALISME, CADRE ET LIMITES MÉTIER

Unités de mesure : Système International

- **Temps :** granularité allant de la seconde au milliard d'années selon la représentation souhaitée
- **Distance entre les objets :** Unité Astronomique (UA = 149 598 000 km) pour les distances entre les objets, km pour les dimensions (rayon et diamètre) des objets
- **Température :** Kelvin
- **Luminosité (photosphère si étoile) / Albedo :** Watt
- **Masse :** kg
- **Volume :** km^3

Etalonnage

Afin de réaliser des comparaisons d'échelles et vu les nombres manipulés, il sera pratique de pouvoir disposer de valeurs intuitives de comparaison, qui seraient donc relatives à un objet.

- **Luminosité des étoiles (photosphère) :** luminosité solaire
- **Masse des étoiles :** masse solaire
- **Rayon des objets (planètes) :** rayon terrestre

Libertés prises par rapport à la réalité

Pour les distances, vu les limites inhérentes à la taille d'un écran, nous perdrions en intuition pour les comparaisons d'échelle, d'où, une série de coefficients de réduction peuvent être injectés via un fichier XML de configuration.

Cela concerne principalement les vues concernant la représentation complète du système solaire.

Un premier modèle fera également abstraction de l'aphélie/périhélie et considérera donc que les trajectoires des orbites sont circulaires et non elliptiques.

Quelques principes astronomiques, physiques et chimiques appliqués :

Température : le rayonnement du corps noir - constante de Planck
Masse : troisième loi de Kepler généralisée
Luminosité : loi de Stefan-Boltzmann
...

3_Détails de l'implémentation

Comme décrit dans la section 2.2. « Découpage en couches », l'accent a notamment été mis sur la réutilisation.
Ceci est donc naturellement reflété au niveau de la structure générale des package :

1. Un canevas général d'une application (principalement la gestion des layout et des menus) a été placée dans le package « be.colorfield » (site Colorfield.be), il s'agit là d'un petit framework entièrement conçu sur mesure, selon mes habitudes de travail.
2. Des API externes ont été utilisées autant que possible, afin de ne pas réinventer la roue.
3. Tous les aspects particuliers à ce projet ont été traités dans le package « org.sse » (site StellarSystemExploration.org).

J'applique ici la convention « web standard » de nomage des package (reverse domain), afin de garantir une absence de conflit qui pourrait être induit par le nomage des classes.

1. ENVIRONNEMENT DE TEST

En natif, il existe un package de test : FlexUnit, qui permet des tests unitaires, assertions, ...

L'enjeu de la 3D web étant les performances, j'ai plutôt travaillé ici sur des tests hardware exploitables durant le temps d'exécution.
Away3D dispose d'un outil d'analyse de performances, accessible via un clic droit.

Pour les autres parties de l'application, une application sur mesure a été utilisée pour mesurer les performances.
Voir également le point 4.5 Mesure et gestion des performances.

Enfin, j'ai crée deux petits utilitaires de test / debug
qui permettent :

1. De parcourir recursivement la DisplayList d'une application (ou donc d'un enfant - conteneur - de la DisplayList Pour ces notions, voir la rubrique 4.3.1. Introduction au langage ActionScript 3).
2. De mesurer la vitesse d'exécution d'un algorithme.

2. STRUCTURE COMMUNE D'UNE APPLICATION

1. Classes liées aux modèles

ActionScript 3 est un langage essentiellement orienté présentation et ne dispose en natif que de deux structures de données composites : Array et Vector.

Je me suis basé sur le travail de Michael Baczynski pour le support de la sémantique des types abstraits de données communément rencontrées en Java (p.ex. Stack, Queue, ...).

2. Classes liées aux vues

Une partie du travail a été délégué aux classes écrites par Dale Sattler : gestion du layout fluide et du redimensionnement bitmap sans dithering (effet «d'escalier» des pixels) pour le fond d'écran. J'y ai ajouté la couche nécessaire à la gestion des menus, des écrans et de la coexistence simultanée des espaces de coordonnées 2D et 3D.

Gestion des menus et des écrans associés

Un fichier XML détient toutes les informations liées au plan de l'application :
des « menuItem » sont composés d'attributs, notamment le nom de la classe à instancier avec ses paramètres.

Chaque noeud est cliquable (s'il comporte l'attribut booléen «clickable»), il mène alors vers le premier élément feuille.
A chaque feuille correspond un objet Screen géré par la classe ScreenManager.

Structure du fichier mainMenu.xml :
Exemple d'un menu comportant un sous-menu.

```
<?xml version=»1.0» encoding=»utf-8» standalone=»yes»?>
<mainMenu>
      (...)
      <menuItem label=»Voyager» subLabel=»Sonder le système
solaire» description=»...» clickable >
            <menuItem label=»Notre système»
subLabel=»Fiches techniques» screenClass=»Screen02PlanetsDe
tails»  clickable />
            <menuItem label=»Perspective» description=»Une
première approche» screenClass=»Screen02PlanetsPerspective»
clickable />
      </menuItem>
      (...)
</mainMenu>
```

La responsabilité de la gestion des menus est assurée par quatre classes :

Celles-ci étant relativement classiques et peu intéressantes au niveau algoritmique ou conception, je m'attarderai plutôt sur la gestion des écrans, ci-après.

be.colorfield.ui.control.MainMenu
Comportement du menu principal (mise en évidence de la rubrique courante, ouverture d'un sous menu s'il y en a un, ...).
Etend Sprite.

be.colorfield.ui.control.MainButton
Comportement des boutons du menu principal
(visibles par défaut et fixes). Etend Sprite.

be.colorfield.ui.control.SubButton
Comportement des boutons du sous-menu
(non visibles par défaut et animés), étendent MainButton.

be.colorfield.events.ButtonEvent
Evénement dispatché par un Button.
Etend Event et permet de définir une propriété complémentaire à celui-ci (l'écran à appeller, le lien, ...).

Chaque Button, quel qu'il soit, faisant partie d'un Menu, pourra également exploiter ce principe,
p.ex. org.sse.ui.control.SubMenuGoogleMaps.

La responsabilité de la gestion des écrans est assurée par trois classes et une interface :

be.colorfield.ui.layout.MainGUI

Gestion du template de layout (éléments invariants d'un écran à l'autre p.ex. menu, fond de l'application, ...)

Méthode private createTemplate
Instanciation des invariants du template graphique et passage des références de ceux-ci au gestionnaire de layout fluide assurant un positionnement similaire aux LayoutManager de Java Swing (North, Center, ...). N'est appelée qu'une seule fois par application. La classe MainGUI est donc un Singleton.

Méthode publique displayTemplateBackground
Affichage des invariants du template devant figurer à l'arrière-plan d'un écran (bas de la pile d'affichage).

Méthode publique displayTemplateForeground
Affichage des invariants du template devant figurer à l'avant-plan d'un écran (haut de la pile d'affichage).

Méthode publique removeAllElements
Vide la pile d'affichage.

be.colorfield.ui.screen.IScreen :

Interface définissant le comportement d'un écran.
Cf. ScreenAbstract pour plus de détails.

```
public interface IScreen
{
    function hasElements():Boolean;
    function buildElements():void;
    function addElement(dObj:DisplayObject,
                        options:LayoutVO):void;
    function removeElement():void;
    function displayElements():void;
    function postDisplayOperations():void;
}
```

be.colorfield.ui.screen.ScreenManager

Stocke une référence à l'écran courant et affiche celui-ci.

Méthode public displayScreen
Reçoit un objet implémentant IScreen.
Si l'écran courant est différent de l'écran passé en paramètre, les méthodes suivantes sont appellées :
MainGUI.removeAllElements();
MainGUI.displayTemplateBackground();
IScreen.displayElements();
MainGUI.displayTemplateForeground();

be.colorfield.ui.screen.ScreenAbstract

Classe abstraite définissant le comportement d'un écran.
On considère que chaque élément est un DisplayObject et qu'il comporte une série d'options de layout via un value object (LayoutVO) qui sera ensuite passé au gestionnaire de layout.

Chaque écran agit alors comme un conteneur disposant de sa propre DisplayList, ce qui reste en parfait accord avec le système d'affichage d'ActionScript (différenciation de l'instanciation et de l'affichage).

Méthode abstraite buildElements
Instancie les objets constituant l'écran

Méthode concrète addElement
Ajoute un objet à la liste d'affichage (reçoit le DisplayObject et ses options d'affichage).

Méthode concrète removeElement
Supprime un objet de la liste d'affichage.

Méthode concrète displayElements
Affiche tous les objets présents dans la liste d'affichage.

Méthode abstraite postDisplayOperations
Opérations qui ne peuvent être appellées qu'une fois les objets affichés à l'écran (= dans la DisplayList principale, de Stage).
Ceci peut être par exemple l'accès à la propriété stage.

L'application cliente (qui crée des écrans)
a donc deux possibilités, soit :
1. Sous-classer la classe ScreenAbstract.
2. Soit implémenter ses propres comportements.

3. Classes utilitaires

Outre les classes utilitaires de debug et de test présentées dans la section 3.1, j'ai créé quelques classes I/0 permettant de charger des fichiers (XML, images, swf, ...)

be.colorfield.net.LoaderXML
be.colorfield.net.LoadExternalAsset

Ces deux classes fonctionnent d'une manière similaire :
Une méthode reçoit en paramètre le chemin d'accès et le nom du fichier, gère les erreurs s'il y en a et dans le cas contraire dispatch un événement (qui est écouté par la classe cliente) lorsque le contenu du fichier est disponible.

D'autres petits utilitaires sont également créés au fur et à mesure des besoins afin de se constituer une « boîte à outils », p.ex.

be.colorfield.utils.FileUtils
Récupération de l'extension d'u fichier à partir de son nom, ...

3. GESTION DU PROJET : LE CMS DRUPAL

Le CMS open source Drupal a été utilisé pour deux objectifs, réalisés par le site internet http://ei.hypercube.be :

1. Présentation du projet

Il s'agit d'un objectif accessoire mais qui s'avère assez pratique, un rapport mis à jour au fur et à mesure de l'élaboration du projet est donc consultable en ligne.
Le type de donnée natif «Livre» a été utilisé afin de chapitrer le rapport et établir une navigation intuitive.

2. Gestion du projet

Vu que les périodes de recherche interrompaient parfois de manière durable les périodes d'intégration, j'ai constaté qu'il était aisé de perdre le fil rouge dans ces conditions.

Afin de remédier à cela, deux types de données ont été créés :

Le type «Module»

Celui-ci décrit une portion atomique (= pouvant exister seul) du système construit et est composé des données suivantes :

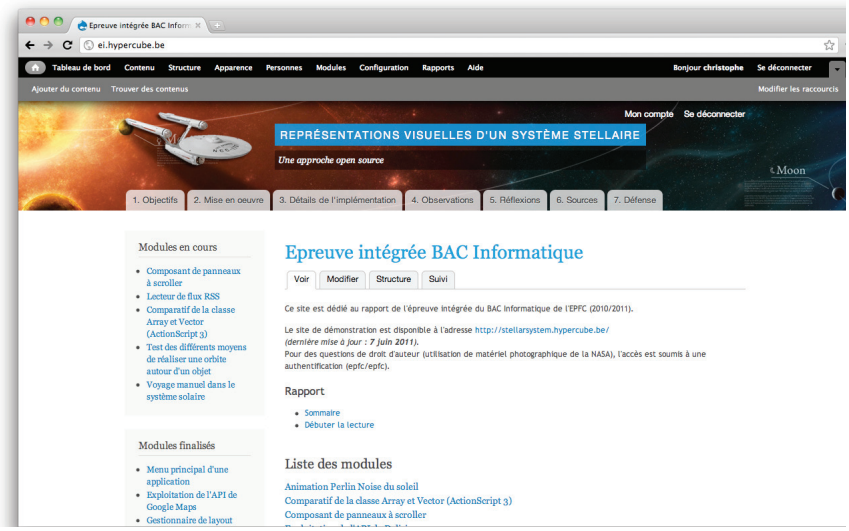
- un titre
- une description
- un statut (A réaliser, En cours, Finalisé)
- une catégorie (Structure, Système stellaire, Test)
- une todo list
- les portions de code relevantes
- un lien vers le module
- de 1 à n références vers le(s) module(s) parents ou frères.

Le type «Mise à jour»

Il décrit une mise à jour conséquente, pouvant porter sur plusieurs modules (p.ex. changement d'API, ...)

Contient

- un titre
- une description
- une date
- de 1 à n références vers les modules concernés.



La page d'accueil contient plusieurs vues (requêtes SQL générées via une interface graphique) de ces types de données

- Les modules en cours
- Les modules finalisés
- Les 5 dernières mises à jour

- Les todo list de chaque module
- Les modules à réaliser

Ce CMS pourrait faire l'objet d'une épreuve intégrée à lui seul, pour plus d'informations sur ce sujet, cf. la rubrique dédiée à Drupal sur mon blog :
http://www.colorfield.eu/category/cms/drupal/

4. DÉTAIL DE QUELQUES MODULES

Cette section détaille quatre modules hors du framework commun. Les deux premiers concernent des applications sur le thème de l'astronomie, les deux suivants ont trait à la création de composants graphiques sur base du design pattern MVC.
Le code de chaque module se retrouve en téléchargement (annexe).

1. Voyage (manuel) dans un système stellaire

Objectif
Permettre de modéliser tout type de système stellaire à partir d'un fichier XML et de textures Bitmap (p.ex. système d'étoiles binaire avec exoplanètes). L'intégralité du fichier XML est disponible en annexe.

API utilisées
Papervision3D, Google Map Planetary

Méthodologie de conception
Utiliser une modélisation suffisament souple pour pouvoir introduire plus de complexité ensuite (p.ex introduction de la notion de gravité pour réaliser les rotations, utilisation du même modèle pour réaliser un voyage automatique, une présentation incluant les satellites ou permettant un niveau de détail inférieur, ...).

Particularités techniques
Camera
La classe SpringCamera a été utilisée pour simuler le voyage dans le système solaire, contrôlé par l'utilisateur.

Shader
La classe GouraudMaterial combiné à une source d'éclairage (localisée au niveau du soleil...) permet un rendu réaliste de l'albedo, ce qui permet de visualiser l'evolution du terminateur lors de la rotation locale de chaque objet (révolution).

Microphone

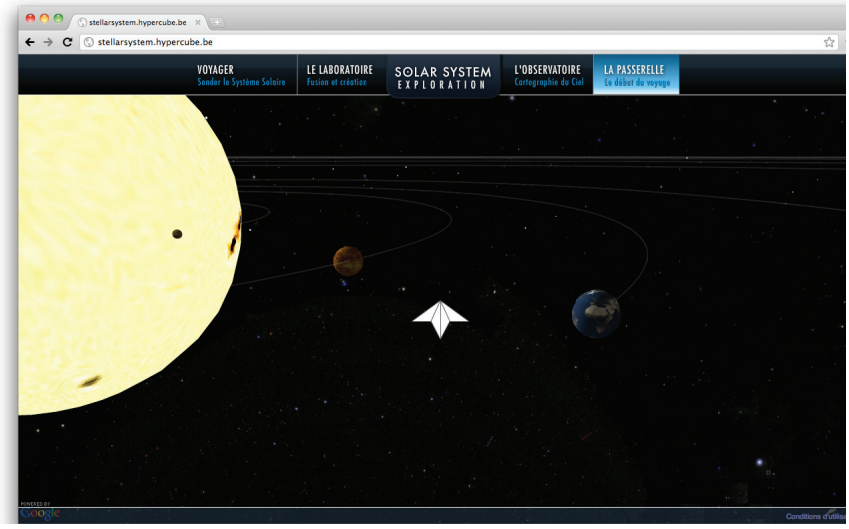
Il est possible d'activer le mode microphone afin d'effectuer un «voyage dans le temps» (inverser l'orbite des objets en soufflant dans le microphone).

Google Map

En toile de fond, une carte de Hubble Deep Space Field a été ajoutée. La fonction de callback onRenderTick de Papervision 3D a été adaptée afin de modifier la longitude lors du déplacement du vaisseau.

UserInputHandler

Cette classe permet de centraliser toutes les entrées de l'utilisateur, chaque vue interrogera cette classe via des propriétés public static.



Diagrammes de classes

La modélisation UML de ce module décrit avec détail

- le modèle (diagramme « Représentation de la matière »)
- les vues et les controlleurs (diagramme « Voyage manuel »)

Ceux-ci sont accessibles en annexe.

Utilisation

Utiliser les touches fléchées pour se déplacer.
Les touches P et L permettent de se déplacer vers le haut / bas par rapport au plan de l'écliptique.
La touche M permet d'activer le mode microphone.

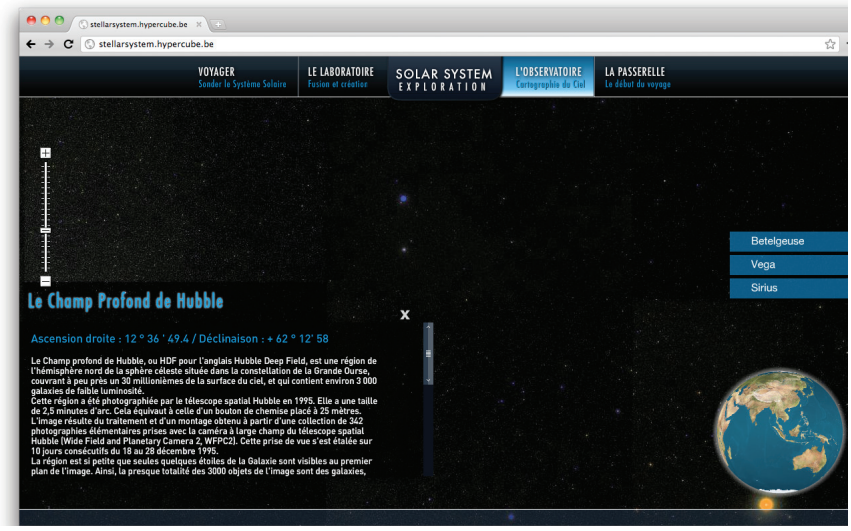
Extension en cours

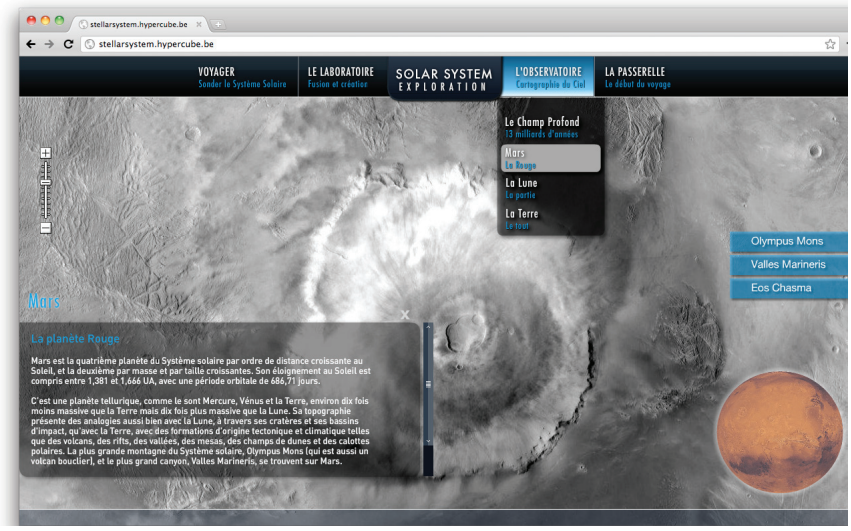
Modification du type de caméra (première personne, troisième personne, ...) via les touches du clavier numérique.

Extensions possible

- Import d'un fichier externe pour la représentation du vaisseau
- Présentation de chaque planète au clic (+ détail des satellites)
- Introduction de la gravité pour l'orbite des objets
- Voyage automatique (calcul des catapultes gravitationnelles)
- Gestion des collisions entre les objets
- Sélection permettant d'atteindre un objet et de se synchoniser sur son orbite
- Les anneaux de Saturne

Utilisation des API Google Map Planetary et Away3D





Recherche préalable

Test des différents moyens de réaliser une orbite autour d'un objet:

1. Comparatif de l'utilisation d'un cercle et d'un cylindre pour visualiser le tracé de l'orbite

Sur une même base (p.ex. représentation d'un atome de carbone, le cercle étant utilisé pour tracer les orbites des électrons), j'ai comparé l'utilisation d'une classe écrite sur mesure Cercle, par rapport à l'utilisation d'une «primitive» Cylinder d'1 pixel de hauteur.

la classe Cercle génère 240 triangles
la classe Cylinder génère 816 triangles
Nous retenons donc Cercle.

2. Comparatif des méthodes pour réaliser une orbite d'un objet autour d'un autre :

1. Rotation d'un conteneur (DisplayObject3D) de l'objet et de son orbite: on crée le tracé de l'orbite, on positionne l'objet sur celui-ci, on réunit les deux dans un conteneur 3D et on fait tourner ce dernier (donc l'ensemble).
2. Utilisation d'une sinusoïde

La dernière méthode s'est avérée être la plus efficace, vu qu'on ne fait tourner que l'objet (et non son conteneur + le tracé).

Méthode 1 :
(extrait...)

```
pivot = new DisplayObject3D();
var lineMaterial:LineMaterial = new
LineMaterial(color,0.1);
orbit = new Circle3D(lineMaterial,moonDistance);
orbit.rotationX = ECLIPTIC; // on définit le plan
moon = new Sphere();
moon.x = distance; // on positionne une fois pour toute
pivot.addChild(orbit);
pivot.addChild(moon);
```

La fonction de callback onRenderTick effectue alors

```
sphere.localRotationY += LOCAL_ROTATION;
do3d.yaw(revolutions[o]*SPEED_REVOLUTION_COEFF);
```

(en bref, on pourrait retenir l'essentiel comme ceci...)

```
earth = new Sphere();
moon = new Sphere();
pivot = new DisplayObject3D();
pivot.addChild(moon);
```

Callback

```
pivot.yaw(1/3); // la lune tourne autour de la terre
moon.yaw(1); // la lune tourne autour de son axe
earth.yaw(1); // la terre également
```

Méthode 2 :

```
earth.rotationY -= simulationRate;
moon.rotationY += simulationRate * 0.1;

moon.x = Math.cos(earthRotation) * moonDistance;
moon.z = Math.sin(earthRotation) * moonDistance;
moon.y = Math.sin(earthRotation) * moonEllipse;
earthRotation -= (simulationRate * 0.1) / 180 * Math.PI;
```

2. Animation Perlin Noise du soleil

Objectif

Obtenir une animation comportant un rendu réaliste d'une activité solaire.

API utilisée

Away3D

Recherche préalable

Nous pouvons nous inspirer de l'algorithme écrit en C par Ken Perlin (Perlin Noise) http://mrl.nyu.edu/~perlin/doc/oscar.html.

L'algorithme permettant de générer un bruit de Perlin interpole et combine des fonctions de bruit aléatoire individuelles (appelées octaves) en fonction unique qui génère un bruit aléatoire qui semble naturel. Tout comme les octaves musicales, la fréquence de chaque fonction d'octave est doublée par rapport à celle qui la précède. Le bruit Perlin est décrit comme étant une « somme de bruit fractale » car il combine plusieurs ensembles de données de bruit avec différents niveaux de détails.

Particularités techniques

Du point de vue de l'implémentation, pour la création de la texture initiale, nous allons utiliser un objet natif BitmapData (flash.display.BitmapData) et appeler sa méthode perlinNoise().

Cet objet sera ensuite utilisé comme Material à plaquer sur la sphère via un BitmapMaterial (Away3D).
Afin d'animer le bruit de Perlin, on passera un tableau d'Offset modifié lors de chaque appel d'un Timer.
Nous ne réalisons pas l'animation du bruit de Perlin dans la fonction de callback du rendering 3D afin de dissocier les deux vitesses d'animation (animation du bruit de Perlin par le Timer et animation de la rotation locale de la sphère par la fonction de rendering).

Pour la modification des couleurs, un cercle 2D sera superposé avec un BlendMode (mode de fusion de deux objets graphiques).
A noter qu'il est tout à fait possible de réaliser l'inverse (la sphère qui réalise un BlendMode avec le cercle 2D, mais cela nuirait aux performances de l'application vu que le calcul de la superposition serait plus important).

Pour la finition on représente la photosphère via une combinaison de deux instances de filtres de rayonnement passés via un array à la méthode setter filters du DisplayObject.

<u>BitmapData</u>
Constructeur BitmapData()
public function BitmapData(width:int, height:int, transparent:Boolean = true, fillColor:uint = 0xFFFFFFFF)

Paramètres de la méthode perlinNoise()
public function perlinNoise(baseX:Number, baseY:Number, numOctaves:uint, randomSeed:int, stitch:Boolean, fractalNoise:Boolean, channelOptions:uint = 7, grayScale:Boolean = false, offsets:Array = null):void

```
// initialisation
offset0:Point = new Point();
offset1:Point = new Point();
bmp.perlinNoise(PERLINNOISE_BASE,
                PERLINNOISE_BASE,
                N_OCTAVES,
                Math.random(),
                true,true,2,true);

// à chaque appel du Timer
offset0.y +=  OFFSET0_Y_DELTA;
offset0.x +=  OFFSET0_X_DELTA;
offset1.y -=  OFFSET1_Y_DELTA;
offset1.x -=  OFFSET1_X_DELTA;

bmp.perlinNoise(PERLINNOISE_BASE, PERLINNOISE_BASE, N_OC-
TAVES, Math.random(), true, false, 2, true,
[offset0, offset1]);
```

Alternatives possibles

Il existe une nouvelle manière de traiter manipulations Bitmap et les filtres : Pixel Bender.
Celle-ci est réputée plus performante et a été portée récemment afin d'être exploitable via les API 3D :
http://labs.adobe.com/technologies/pixelbender3d/

3. Composant de panneaux à scroller

Objectif

Je comptais déléguer cette tâche à un composant existant, jusqu'à ce qu'il s'avère qu'il ne remplissait pas la facilité d'utilisation souhaitée. **Cela peut être considéré comme un contre-exemple quant à l'utilisation de l'open source** : pour de petites fonctionnalités légères, il vaut parfois mieux écrire le composant selon ses nécessités.

Méthodologie de conception

Ce fut donc l'occasion d'écrire un composant simple en séparant les responsabilités selon le pattern MVC.

Particularités techniques

La philosophie d'implémentation reste la même que celle utilisée dans la totalité du projet : une partie générique, réutilisable et définie dans le package be.colorfield.ui.box, puis une partie spécifique définie dans le package org.sse.ui.container.

ScrollingPanelModel
ValueObject conservant le titre

ScrollingPanelView
Classe étendant une définition créée sous l'environnement graphique de Flash (étend donc MovieClip).

ScrollablePanels
Contrôleur et conteneur logique des ScrollingPanelView

L'application cliente
1. Instancie un Vector de ScrollingPanelModel, peuplé par des objets graphiques étendant Sprite.
2. Passe ce Vector à une instance de ScrollablePanels

D'un point de vue spécifique à cette application, on va passer à ScrollingPanelView des objets PlanetDetail (une sous-classe de Sprite). Tout objet étendant Sprite étant accepté.

```
var sp:ScrollingPanels = new ScrollingPanels();

var panels:Vector.<ScrollablePanelModel> = new
Vector.<ScrollablePanelModel>;

// le tout devrait être instancié dans une boucle,
// depuis un fichier XML p.ex.
// par concision, on donne un exemple de base
var panel1:ScrollablePanelModel = new
ScrollablePanelModel();
panel1.setTitle(«Test 1»);
panel1.setContent(new PlanetDescription());
panels.push(panel1);

var panel2:ScrollablePanelModel = new
ScrollablePanelModel();
panel2.setTitle(«Test 2»);
panel2.setContent(new PlanetDescription());
panels.push(panel2);

sp.createPanels(panels);
addChild(sp);
```

4. Lecteur de flux RSS

Objectifs

1. Récupérer le flux RSS de l'ESA (Agence Spatiale Européenne)
2. Notifier l'utilisateur des mises à jour de l'application sur l'écran d'accueil

Particularités techniques

1. Récupération du flux RSS de l'ESA
Dans le premier cas, nous nous trouvons devant un **problème de sécurité au niveau réseau** :
Le sous-domaine stellarsystem.hypercube.be tolère la récupération de flux par d'autres domaines grâce à la présence d'un fichier XML crossdomain présent à sa racine (http://stellarsystem.hypercube.be/crossdomain.xml), toutefois, l'équivalent n'existe pas sur le domaine esa.int (erreur http 404 lors du la tentative de récupération de ce fichier par le sandbox du Flash Player).

Nous serons donc obligé de récupérer physiquement le flux RSS sur l'espace disque de l'hôte virtuel dédié à notre domaine.
Une tâche cron sera donc écrite à cet effet.

Notons toutefois que dans le cas du déploiement via AIR (Adobe Integrated Runtime), ce problème n'est pas rencontré vu qu'il s'agit plus d'une application web, mais bien d'une application desktop.

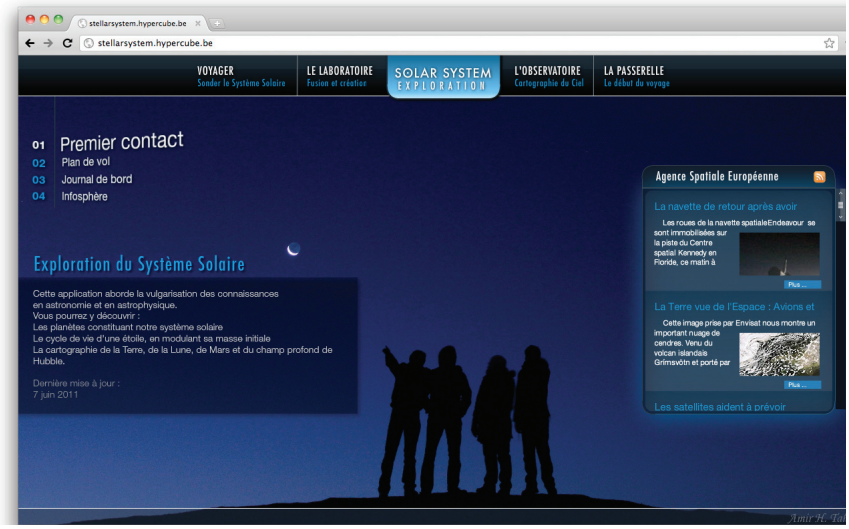
2. Notification des mises à jour de l'application
Dans ce cas, la sécurité est entièrement contrôlée, vu que tous les sous-domaine hypercube.be peuvent accéder au sous-domaine ei.hypercube.be (caractère joker *.hypercube.be) ; on souhaite également que les domaines www.colorfield.eu et www.colorfield.be puissent accéder (cf. http://ei.hypercube.be/crossdomain.xml)

Une vue avait été créée via Drupal afin de présenter sur le site de gestion (page d'accueil), les derniers modules finalisés. Nous allons naturellement exploiter celle-ci en modifiant uniquement l'output (on ne souhaite plus un rendu html, mais une sortie XML suivant la norme RSS). Celle-ci est simplement rendue accessible via le chemin http://ei.hypercube.be/?q=modules_finalises.

Pour ces deux cas, une classe RSSModel se charge de la gestion des données, tandis que deux vues permettent un rendu différent:

Pour le RSS de l'ESA, une vue complète a été définie (text, images, habillage graphique, ...) alors que pour les mises à jour il suffisait de faire un append de text.

Le diagramme UML se retrouve en annexe.



4_Observations

1. INFORMATIQUE ET GRAPHISME

Plusieurs domaines de l'informatique se sont spécialisés dans la conception graphique.
Prenons quelques exemples selon le type d'application :
Concernant les **applications desktop** dans le domaine de la recherche graphique, on peut citer **Processing** (basé sur la JVM) ou **NodeBox** (utilisant Python).
Du côté des **applications Web**, on pourra trouver deux grandes tendances : les formats standards et ouverts tels que **HTML 5, jQuery, CSS 3** versus les formats propriétaires tels que la **Flash Platform**. On remarquera que même Google s'y perdra et publiera durant plusieurs années pour les deux types de formats (prenons l'exemple de la famille d'API Google Maps, ou des Charts).

Considérons maintenant quelques exemples selon la destination (les deux existant en desktop et web) :

- La **mise en forme de publications** : on peut citer le langage **Latex**, le format **ePub**, ...
- Les **jeux vidéos** desktop sont largement portés par **C++**, (je pense notamment aux multitudes de librairies telles que les moteurs de particules et autres applications de la physique en général), mais existent de plus en plus sous la forme d'applications réseau, impliquant d'écrire des applications côté serveur de plus en plus performantes (C++ encore...).

Ce ne sont que quelques exemples en guise d'introduction, me laissant à penser qu'il y a de beaux jours pour cette mixité de la recherche. Pour présenter cette infime partie de ce vaste domaine, j'ai choisit la Flash Platform, et en particulier le langage ActionScript 3.

2. ACTIONSCRIPT 3

1. Introduction au langage ActionScript

Présentation générale et historique

ActionScript est un langage essentiellement dédié au front end, à la présentation visuelle. Il existe toutefois des solutions backend (CMS,...) implémentées en ActionScript, mais elles sont plutôt râres.
Le langage n'est donc pas vraiment orienté vers les types de données abstraits. Si on souhaite une implémentation d'une liste, d'un arbre, d'une file de priorité, ... il faudra se tourner vers des solutions externes (cf. implémentation réalisée par Michael Baczynski).

ActionScript 1 était entièrement séquentiel et procédural, avec un typage très faible (à la manière de PHP). Les scripts étaient intégrés directement sur les instances des MovieClip et Button ou ajoutés dans les frame. Cela a amené à des pratiques extrêmement douteuses et a largement contribué à une mauvaise réputation de Flash.

ActionScript 2 a ajouté un typage plus strict, cependant, sa déclaration était laissée à la liberté du programmeur. L'avantage était une vérification du type à la compilation. Tout comme PHP 4, on a constaté un support de la POO, mais assez bancal vu que la version 2 se basait principalement sur la première. On a donc vu fleurir d' « ingénieux » mélanges de procédural et d'orienté objet.

Lors de la reprise de la société Macromedia par Adobe (version 8 du lecteur Flash), **ActionScript 3** a été entièrement refondu selon les spécifications ECMAScript (standard de Mozilla). Le résultat est un langage entièrement orienté objet, avec un typage strict.

Notons toutefois qu'il a fallu attendre la version 10 du lecteur Flash pour voir la prise en charge de structures telles que la classe Vector, permettant au programmeur de déclarer une ensemble composé d'un seul type d'objet. Auparavant, seul la classe Array existait (il s'agissait en fait d'une liste).

A partir de la version 3 du langage, un IDE dédié uniquement au développement (et non à la conception graphique) a vu le jour : Flex (basé sur Eclipse). Sa version 2 fut renommée Flex Builder et la 4 Flash Builder ... de quoi perdre le «newbie».

Les objets destinés à l'affichage

La classe racine des objets destinés à être affichés à l'écran (faisant donc partie de la DisplayList) est DisplayObject, dont voici l'arbre d'héritage :

DisplayObject :
→ Shape, Bitmap, Video, InteractiveObject, AVM1Movie, MorphShape, StaticText

InteractiveObject :
→ SimpleButton, DisplayObjectContainer, TextField

DisplayObjectContainer :
→ Stage, Sprite, Loader

Sprite :
→ MovieClip

Stage est le conteneur principal d'une application. Celle-ci contient une DisplayList.
Chaque objet ajouté à la DisplayList dispose d'une propriété stage permettant d'accéder à celle-ci (pour peu que l'objet se trouve dans la DisplayList = soit présent à l'écran).

La différence fondamentale entre la classe Sprite et MovieClip vient de l'historique de Flash : celui-ci était initiallement très orienté animation «manuelle» via un système de frame. Un MovieClip dispose d'une ligne du temps, tandis qu'un Sprite non.

Le modèle événementiel

L'utilisation d'événements est omniprésente.
Le modèle se base sur le design pattern Observable - Observer.

Une description de ce modèle est réalisée au point :
4.5. Mesure et gestion des performances - Gestion des écouteurs d'événements.

La gestion des erreurs

Le mécanisme est clairement inspiré des langages tels que Java :

```
try{
  // code qui pourrait générer une erreur
} catch (err:Error)
{
  // code qui réagit à l'erreur
}
finally{
  // code exécuté dans tous les cas
  //(qu'il y ait une erreur ou non
  // peut éventuellement réaliser un nettoyage après l'erreur
}
```

2. Quelques particularités du langage

La portée des variables

Une variable i déclarée dans une instruction `for(var i:int;i<length;++i)` sera visible hors de la boucle.

Gestion de la DisplayList

La DisplayList est implémentée sous la forme d'une pile, un moyen efficace de supprimer tous les enfants d'un conteneur pourrait être le suivant :

```
var i:int = container_mc.numChildren;
while( i -- )
    container_mc.removeChildAt( i );
```

Définition d'un constructeur par défaut, avec des paramètres

```
public function ScrollingBox(scrollBarHighlight:int
                             = 0xcccccc){
    this.scrollBarHighlight = scrollBarHighlight;
}
```

Dans le même ordre idée, il arrive de vouloir déclencher une méthode de deux manières : en réponse à un événement ou via un appel manuel. Plutôt que de créer une méthode stub, on écrira

```
private function maMethode(e:Event = null):void{
 // faire quelque chose
}
```

On pourra alors réaliser les deux types d'appels suivants :

```
this.addEventListener(Event.ADDED_TO_STAGE, maMethode);
this.maMethode();
```

Accesseurs et mutateurs

Une syntaxe particulière permet d'accéder / muter les données private via la notation pointée comme s'il s'agissait d'une propriété publique :
p.ex. la classe cliente appelera

```
xmlStream = xmlLoader.xmlData;
```

au lieu de

```
xmlStream = xmlLoader.getXmlData();

public function get xmlData ( ):XML{
 return xmlStream;
}
```

Traitement de XML

ActionScript 3 suit la spécification ECMAScript E4X (ECMAScript for XML) en adoptant le traitement des flux XML via une notation pointée, et non via le DOM (Document Object Model). Il en résulte une utilisation intuitive, comme une extension du langage.

Syntaxe JavaDoc

La syntaxe JavaDoc pour les commentaires est supportée. Le générateur ASDOC produira une sortie de ce type :
http://www.colorfield.eu/files/epfc/prdv/snake/asdoc/index.html

L'échelle à 9 découpes

La propriété de la classe Sprite, Scale9Grid, permet de conserver le radius d'un composant via une échelle à neuf découpes.
Ceci permet par exemple de redimensionner un composant disposant de coins arrondis sans modifier le rayon de ses coins.

Absence de classe abstraite

Aucun mécanisme de gestion des classes abstraites n'est accessible en ActionScript 3.

Le programmeur devra donc prévoir de documenter via ASDOC le statut d'une classe abstraite.

Les méthodes abstraites ne pouvant être déclarées comme telles, devront explicitement lancer une exception.

Exemple d'usage : be.colorfield.ui.ScreenAbstract

2.3. Etude d'un cas : le design pattern Singleton

Le problème : le constructeur ne peut être private

L'implémentation classique que l'on pourrait réaliser en Java ne fonctionne donc pas:

```
class Singleton{
 private static const INSTANCE:Singleton = new Singleton();
 private function Singleton(){}
 public static function getInstance():Singleton{
   return Singleton.INSTANCE;
 }
}
```

Une solution : créer une classe privée

```
public class Singleton{
 private static const INSTANCE:Singleton =
                      new Singleton(new PrivateClass());

 public function Singleton(pvt:PrivateClass){
 }
 public static function getInstance():Singleton{
   return Singleton.INSTANCE;
 }
}

internal class PrivateClass{
 public function PrivateClass(){}
}
```

Théoriquement, cela devrait fonctionner...

TypeError: Error #1115: ::SingletonPrivateClass n'est pas un constructeur.
at be.colorfield.ui.layout::MainGUI$cinit() at global$init() at Main/loaderInit()

Apparement le class loader n'a pas encore fait son travail ?
Il faudra donc transformer l'implémentation ci-dessus :

```
public class Singleton{

 private static var INSTANCE:Singleton;
 public function Singleton(pvt:PrivateClass){
 }
 public static function getInstance():Singleton{
  if(Singleton.INSTANCE == null){
    Singleton.INSTANCE = new Singleton(new PrivateClass());
  }
  return Singleton.INSTANCE;
 }
}

internal class PrivateClass{

}
```

Usage :

Il est donc impossible d'instancier via `new Singleton();` (plus de constructeur par défaut), ce qui provoquerait une erreur à la compilation.

Si nous essayons d'instancier via `new Singleton(new PrivateClass());`, cela provoquera une nouvelle erreur à la compilation, vu que la classe privée ne peut être accédée que via la classe Singleton.

Une implémentation correcte (et «classique») sera donc :

```
var mySingleton:Singleton = Singleton.getInstance();
```

2.4. Les threads

Contrairement à Java, nous ne devons pas gérer l'aspect thread safe, vu que «While the underlying Flash Player and Operating System use threads, the execution model for Actionscript does not.». Source : http://blogs.adobe.com/aharui/2008/01/threads_in_actionscript_3.html

Notons toutefois que si nous souhaitons donner l'illusion d'animer deux composants simultanément en Java, il n'y a pas d'autre solution que de passer par des threads. ActionScript, avec son modèle d'exécution par «frame» s'apparente alors plus à un ordonancement sans gestion de file de priorité (chaque frame est responsable de représenter une partie d'une animation).

2.5. Performances

Quelques tests de performances pour nous indiquer quels types de données utiliser.

Les types simples

Les types numériques : int et uint

- Le type **int** permet de représenter un nombre entier de 32 bits
- Le type **uint** également mais non signé
- Le type **Number** permet de représenter un nombre décimal de 64 bits

Une hésitation peut donc subvenir lors de l'utilisation du type int ou uint. La littérature nous déconseille d'utiliser le type uint car il s'avère en général beaucoup plus lent lorsqu'une opération mathématique est effectuée.

Vérifions ceci :

```
const LNG:int = 5000000;

var startTimer:Number = getTimer();
for(var i:int = 0; i < LNG;++i)
trace(getTimer() - startTimer); // affiche ~ 60 secondes

var startTimer:Number = getTimer();
for(var i:uint = 0; i < LNG;++i)
trace(getTimer() - startTimer); // affiche ~ 240 secondes

var startTimer:Number = getTimer();
for(var i:Number = 0; i < LNG;++i)
trace(getTimer() - startTimer); // affiche ~ 100 secondes
```

Les types composites
Les classes Array et Vector

La classe Array, existant depuis Flash Player 8 peut être considérée comme une liste, permettant d'accueillir tous types de données.

Le lecteur Flash 10 introduit la classe Vector (spécification ECMAScript 4 de Mozilla). Celle-ci n'accueille qu'une liste d'objets spécifiques.
La signature du constructeur est la suivante:

```
public function Vector(length:uint = 0, fixed:Boolean = false)
```

L'intérêt principal de l'introduction de cette classe est la performance.
(Un autre avantage de la classe Vector est la vérification du type à la compilation).
La preuve par l'exemple :

```
const LNG:int = 2000000;

var data:Vector.<int> = new Vector.<int>(LNG, true);
var startTimer:Number = getTimer();
for(var i:int = 0; i < LNG;++i)
  data[i] = 0;
trace(getTimer() - startTimer); // affiche ~ 60 secondes

//----------------

var data:Array = new Array();
var startTimer:Number = getTimer();
for(var i:int = 0; i < LNG;++i)
  data[i] = 0;
trace(getTimer() - startTimer); // affiche ~ 110 secondes
```

Une fois ce test de base effectué, nous pouvons tester également le fait de créer un vecteur accueillant différents types d'objets

```
var data:Vector.<*> = new Vector.<*>();
```

Même sans spécifier la taille, les performances s'avèrent plus intéressantes qu'avec la classe Array.

2.6. Utilisation du hardware

Flash Player utilise par défaut le CPU et non le GPU, pour des raisons historiques de compatibilité matérielle.

Depuis le Player 10, celui-ci est capable de tirer parti de la puissance de la carte graphique pour optimiser les performances d'affichage des animations Flash. Cela devrait permettre aux fichiers SWF de consommer moins de ressources CPU.
Les fichiers Flash devront cependant être spécialement optimisés pour bénéficier de ces fonctionnalités d'accélération matérielle.
Une carte graphique compatible DirectX 9.0 ou OpenGL 2.0 sera également nécessaire.
Cela ne couvre toutefois pas pour l'instant la prise en charge par les API 3D.

Une évolution récente du Flash Player, prévue pour mi 2011 permet de combler cette lacune.
La nouvelle API Away 3D en tient d'ailleurs compte.
Cela a donc joué dans la sélection du moteur 3D
(cf. observations - sélection du moteur 3D).
http://away3d.com/away3d-adobes-new-3d-api-molehill

Un descriptif du mode de fonctionnement de l'accélération matérielle a été publié par Adobe.

En voici la partie relevante :

Direct and GPU-compositing window modes
Flash Player 10 introduces two window modes, direct and GPU compositing, which you can enable through the publish settings in the Flash authoring tool. These modes are not supported in AIR. To take advantage of these modes, you must enable hardware acceleration for Flash Player.

Direct mode uses the fastest, most direct path to push graphics to the screen, which is advantageous for video playback.

GPU Compositing uses the graphics processing unit on the video card to accelerate compositing. Video compositing is the process of layering multiple images to create a single video image. When compositing is accelerated with the GPU it can improve the performance of YUV conversion, color correction, rotation or scaling, and blending. YUV conversion refers to the color conversion of composite analog signals, which are used for transmission, to the RGB (red, green, blue) color model that video cameras and displays use.

Using the GPU to accelerate compositing reduces the memory and computational demands that are otherwise placed on the CPU. It also results in smoother playback for standard-definition video.

Be cautious in implementing these window modes. Using GPU compositing can be expensive for memory and CPU resources. If some operations (such as blend modes, filtering, clipping or masking) cannot be carried out in the GPU, they are done by the software. Adobe recommends limiting yourself to one SWF file per HTML page when using these modes and you should not enable these modes for banners. The Flash Test Movie facility does not use hardware acceleration but you can use it through the Publish Preview option.

Setting a frame rate in your SWF file that is higher than 60, the maximum screen refresh rate, is useless. Setting the frame rate from 50 through 55 allows for dropped frames, which can occur for various reasons from time to time.

Using direct mode requires Microsoft DirectX 9 with VRAM 128 MB on Windows and OpenGL for Apple Macintosh, Mac OS X v10.2 or higher. GPU compositing requires Microsoft DirectX 9 and Pixel Shader 2.0 support on Windows with 128 MB of VRAM. On Mac OS X and Linux, GPU compositing requires OpenGL 1.5 and several OpenGL extensions (framebuffer object, multitexture, shader objects, shading language, fragment shader).

You can activate direct and gpu acceleration modes on a per-SWF basis through the Flash Publish Settings dialog box, using the Hardware Acceleration menu on the Flash tab. If you choose None, the window mode reverts to default, transparent, or opaque, as specified by the Window Mode setting on the HTML tab.

3. LES API 3D EN ACTIONSCRIPT 3

Depuis Flash Player 8, plusieurs développeurs se sont attellés à la création d'un moteur 3D, afin de créer une API totalement inexistante du langage ActionScript. Adobe a finalement intégré une API partielle dans la version de son Player 10. On parle dans ce dernier cas de «2.5D» vu qu'il s'agit de simples plans 2D projetés dans un espace à 3 dimensions.

La multitude de solutions disponibles a nécessité une méthode de sélection :

1. Deux API (Away3D et Papervision3D) ont été retenues en tenant compte des critères suivants : popularité (mesurable par la communauté d'utilisateurs) et documentation disponible (sites, livres, ...).
2. Un comparatif de chaque API a été effectué, principalement sur la base de leurs performances.

Le choix entre les deux API a été assez ardu :

Away3D est plus contemporain (créé en 2007) et se positionne (selon ses développeurs), comme le successeur de Papervision3D (créé en 2005).

Away3D présente globalement de meilleures performances et intègre certaines fonctionnalités liées à la physique, alors que Papervision3D dispose d'une plus vaste documentation et présente certaines fonctionnalités non disponibles chez son homologue.

Away3D 4.0 Broomstick devrait constituer une révolution en matière de 3D Web, vu que les développeurs de cette API et l'équipe d'Adobe ont travaillé de concert afin d'optimiser l'accélération matérielle (cf. point précédent).

Notons qu'il existe également d'autres alternatives ActionScript: Sandy3D, Alternativa3D, ... qui ont toutes pour objet de combler les lacunes de la prise en charge de la 3D par l'API native fournie par Adobe.

Enfin, nous ne pouvons pas passer sous silence les récentes évolutions d'HTML5/CSS3/JavaScript qui se chargent de prendre la « relève » (animation et 3D).

Quelques exemples (utiliser le navigateur Chrome) :

Animation du système solaire (2D)
http://neography.com/experiment/circles/solarsystem/

Animation 3D en HTML 5
http://ie.microsoft.com/testdrive/Performance/Galactic/Default.html

Enfin, l'opinion de Seb Lee-Delisle (« guru» en la matière) laisse présager du futur dans cet article comparant HTML5 et Flash, une polémique très actuelle.
http://sebleedelisle.com/2011/01/html5-vs-flash-the-aftermath/

En voici le résumé :

@seb_ly
Seb Lee-Delisle

Any Flash developers who still think that their work is in any way acceptable to the wider open web community are sadly mistaken. #naconf

Pour les conclusions concernant ce sujet, se référer au point 5. Réflexions.

1. Concepts communs aux API 3D

Que nous utilisions l'API native, Away3D our Papervision3D, nous pouvons dégager plusieurs concepts communs :

Un espace de coordonnées x, y, z :

Une différence de taille :
L'espace de l'API native prend son point d'origine (0,0,0) en haut à gauche de l'écran, tandis que Away3D le prend en bas à gauche !

La vue

View3D (Away3D) ou **ViewPort** (Papervision3D).
Cette classe étend DisplayObject3D et détient la représentation 2D (= l'affichage à l'écran), de la scène.

Le point de fuite
Il s'agit de la projection de perspective : convertir des coordonnées 3D dans un espace de coordonnées 2D.

Le rendering
Le rendu est assuré par une méthode render (Away3D) ou onRenderTick (Papervision3D) appellée sur la vue.

La scène
Scene3D (Away3D et Papervision3D).
Il s'agit d'une version 3D de la classe ActionScript native Stage, qui sera donc le conteneur de niveau supérieur des DisplayObject3D.

La caméra
Camera3D (Away3D et Papervision3D).
Hérite d'Object3D et est un DisplayObject3D. Elle peut donc être déplacée et tournée. Il s'agit du point d'observation dans la scène.

Objets 3D : primitives, modèles, sprites, ...
L'élément racine est **Object3D**, spécialisé par **DisplayObject3D**.
Viennent ensuite les spécialisations suivantes :

- **Vertex** : un vecteur (x, y, z) invisible, mais servant à construire des structures 3D (un triangle est une surface de 3 vertex).
- **Face** (surface) et **Segment** (ligne)
- **Meshes** : collection des vertices (pluriel de vertex)
- **Sprite** : un élément 2D intégré dans un espace 3D
- **Une série de formes primitives :** Plane, Cube, Sphere, ...

Les materials
Ceux-ci peuvent être non «sensibles» à la lumière (WireMaterial, ColorMaterial, ...) ou l'être (WhiteShadingMaterial, BitmapShading-Material, ...) et sont responsables de la texture d'un objet 3D.

Les éclairages
Ceux-ci peuvent être de plusieurs types : PointLight3D, DirectionalLight3D, ...

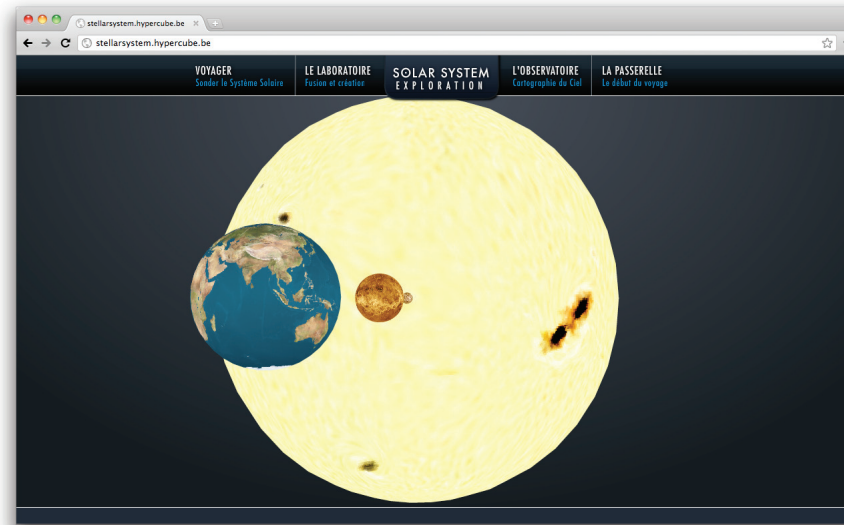
2. Application minimaliste Away3D

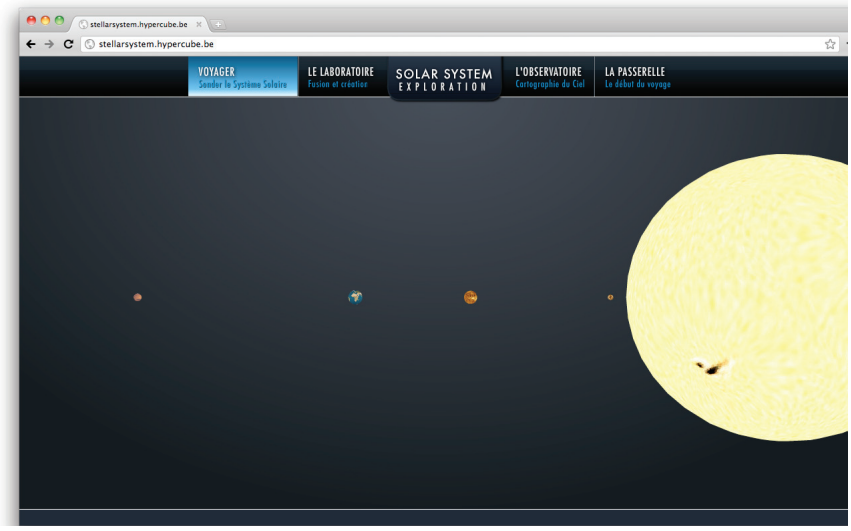
```
package
{
  import away3d.containers.View3D;
  import away3d.events.MouseEvent3D;
  import away3d.lights.PointLight3D;
  import away3d.materials.BitmapMaterial;
  import away3d.materials.WhiteShadingBitmapMaterial;
  import away3d.primitives.Sphere;

  import flash.display.BitmapData;
  import flash.display.Sprite;
  import flash.events.Event;

  /**
   * Dessin d'un cube et d'une sphère
   * avec application d'une texture et d'un éclairage
   */
  public class TestAway3D extends Sprite
  {

  private var view:View3D;
  private var sphere:Sphere;

  public function TestAway3D()
  {
    view = new View3D();
    // positionnement du point de fuite au centre
    view.x = Stage.stageWidth / 2;
    view.y = Stage.stageHeight / 2;
    addChild(view);

    // rendu à chaque frame
    addEventListener(Event.ENTER_FRAME, onEnterFrame);

    // material
    var bmp:BitmapData = new BitmapData(200,200);
    bmp.perlinNoise(200,200,2,Math.random(),true,true);

    var mat:BitmapMaterial = new BitmapMaterial(bmp);

    // -------------------
    // -------------------

    // scene:Scene3D = container root => ~ displayList
    // => addChild / removeChild

    // primitive 1 : cube
    var cube:Cube = new Cube({material:mat});
    view.scene.addChild(cube);

    // gestionnaire de clic
    cube.addEventListener(MouseEvent3D.MOUSE_UP,
                          onClickCube);
```

```
    // --------
    // primitive 2 : sphere
    sphere = new Sphere();
    sphere.segmentsW = 32;
    sphere.segmentsH = 32;
    // si on souhaite que le material soit illuminé
    sphere.material = new WhiteShadingBitmapMaterial(bmp);
    sphere.x = 200;
    view.scene.addChild(sphere);

    var light:PointLight3D = new PointLight3D();
    light.y = 500;
    view.scene.addLight(light);
  }

  private function onEnterFrame(e:Event):void
  {
    sphere.rotationY += 5;
    view.render();
  }

  private function onClickCube(e:MouseEvent3D):void
  {
    // manipulation sur le cube
  }
  } // fin de class
} // fin de package
```

Exemple d'utilisation de cette API : le module « Perspective - Une première approche du système solaire ».

Une application de l'API Away3D utilisant la classe HoverCamera, non disponible en Papervision3D (natif), mais aussi la sélection de différents points de vue par l'utilisateur.





3. Application minimaliste Papervision3D

```
package{

  import org.papervision3d.cameras.Camera3D;
  import org.papervision3d.objects.primitives.Sphere;
  import org.papervision3d.render.BasicRenderEngine;
  import org.papervision3d.scenes.Scene3D;
  import org.papervision3d.view.Viewport3D;

  import flash.display.Sprite;
  import flash.events.Event;

  public class TestPapervision3D extends Sprite
  {
    private var scene:Scene3D;
    private var viewport:Viewport3D;
    private var camera:Camera3D;
    private var renderEngine:BasicRenderEngine;
    private var sphere:Sphere;

    /**
     * Animation d'une sphère
     */
    public function TestPapervision3D()
    {
      scene = new Scene3D();
      camera = new Camera3D();
      sphere = new Sphere();
      scene.addChild(sphere);
      viewport = new Viewport3D();
      addChild(viewport);

      renderEngine = new BasicRenderEngine();
      renderEngine.renderScene(scene, camera, viewport);
      addEventListener(Event.ENTER_FRAME, render);
    }
```

```
    private function render(e:Event):void
    {
      sphere.localRotationY +=1;
      renderEngine.renderScene(scene, camera, viewport);
    }
  } // fin de class
} // fin de package
```

4. Comparatif des deux API

Les emprunts sont nombreux et on constate plus qu'une influence mutuelle, mais plutôt un apport continu. Papervision3D dispose de certaines cameras (SpringCamera p.ex.) alors que Away3D en dispose d'autres (HoverCamera p.ex.), le portage de l'une à l'autre est de tout manière réalisée par des contributeurs puis intégrée au noyau si cela s'avère judicieux.

Au niveau des performances, plusieurs tests ont été réalisés via les outils décrits ci-dessous. Globalement elles restent identiques, vu que les points de comparaisons sont trop nombreux.

5. MESURE ET GESTION DES PERFORMANCES

ActionScript 3

L'optimisation se réalise à plusieurs niveaux :

Mise en cache sous forme de bitmap:
L'API de dessin d'ActionScript permet d'utiliser des vecteurs ou des bitmap.
On utilisera principalement le vecteur : pour les formes, les dessins complexes, ... vu qu'ils peuvent être aggrandis sans perte de qualité, cependant, ceux-ci ont l'inconvéniant d'être consommateurs en temps de calcul.
Il est donc souhaitable, lorsque cela est possible (pas de changement d'état des vecteurs), d'utiliser la mise en cache sous forme de bitmap afin de ne réaliser le calcul qu'une seule fois. Cette propriété des Shape, Sprite et MovieClip s'active simplement en la définissant à true (cacheAsBitmap = true).

Gestion des écouteurs d'événements:
Un écouteur d'événement devrait être explicitement supprimé lorsqu'on n'en a plus besoin.

P.ex., en utilisant l'événement `Event.ADDED_TO_STAGE`, on initialise tous les événements : `monSprite.addEventListener(MouseEvent.CLICK, maFonctionDeCallback)`, puis l'événement `Event.REMOVED_FROM_STAGE` supprime celui-ci : `monSprite.removeEventListener(MouseEvent.CLICK, maFonctionDeCallback)`.

Bien entendu, on supprimera également l'événement `Event.ADDED_TO_STAGE` lui-même, lorsque sa fonction de callback est appellée, mais aussi `Event.REMOVED_FROM_STAGE`.
Cette pratique n'étant pas toujours respectée, l'implémentation minimale permettra au Garbage Collector d'effectuer son travail, en utilisant la propriété **weakReference** : `eventTarget.addEventListener(EventType.EVENT_NAME, eventResponse, useCapture:Booolean, priority:int, weakReference:Boolean)`.

P.ex. : `eventTarget.addEventListener(EventType.EVENT_NAME, eventResponse, false, 0, true)`.

Notons également que du strict point de vue des performances, il vaut mieux, **pour les accès fréquents** (p.ex. entrées au clavier pour diriger un vaisseau spatial) **ne pas passer par la diffusion d'événements** (observable / observer) mais plutôt par une propriété publique statique et que la classe cliente réalise un update sur des objets implémentants une interface commune. L'exemple se retrouve dans le module «Voyage dans le système solaire» avec la classe cliente ManualVoyageView et la classe UserInputController, p.ex. On utilisera par contre un Observable - Observer pour découpler la méthode update du rendu principal.

Papervision3D

L'outil intégré à l'API peut être instancié de cette manière :

```
var stats:StatsView = new StatsView(this.renderer);
addChild(stats);
```

En partant du principe que la classe courante (this) étend BasicView.

Les propriétés suivantes sont alors mesurées :
FPS – Frames Per Second
Tri – Triangles
Sha – Shaded Triangles

Lin – Lines
Par – Particles
COb – Culled Objects
CTr – Culled Triangles
CPa – Culled Particles
FOb – Filtered Objects
Mem – Memory Used

Away3D

Les performances sont mesurables via un outil pré-intégré, accessible via le menu contextuel du Flash Player.

Optimisation commune au deux API

Le clipping et le culling

Il s'agit de deux techniques d'optimisation des performances :

Le **culling** réduit le nombre de calculs du CPU en retirant complètement les polygones se trouvant en dehors de la vue.

Le **clipping** élimine partiellement les polygones qui sont aux limites de la vue.

Optimisation propre à Away3D

Les objets LevelOfDetail, qui diminuent progressivement le détail d'un polygone en fonction de son éloignement.

6. L'ANIMATION EN ACTIONSCRIPT 3

Il existe deux méthode de base pour pouvoir animer (en plus de l'éditeur de mouvement de Flash) :

1. L'événement EnterFrame
2. L'évenement TimerEvent

La différence fondamentale entre les deux étant que l'événement onEnterFrame est lié à la vitesse de lecture globale de l'animation (FrameRate).

Le Timer est complètement indépendant et on pourra donc dédier des événements particuliers pour chaque objet graphique, ce qui simplifie la modulation d'animations telles que les effets de parallaxe (http://fr.wikipedia.org/wiki/Parallaxe).

Un livre entier a été écrit sur le sujet de l'animation ActionScript 3, sans utiliser de framework (http://www.friendsofed.com/book.html?isbn=1590597915).

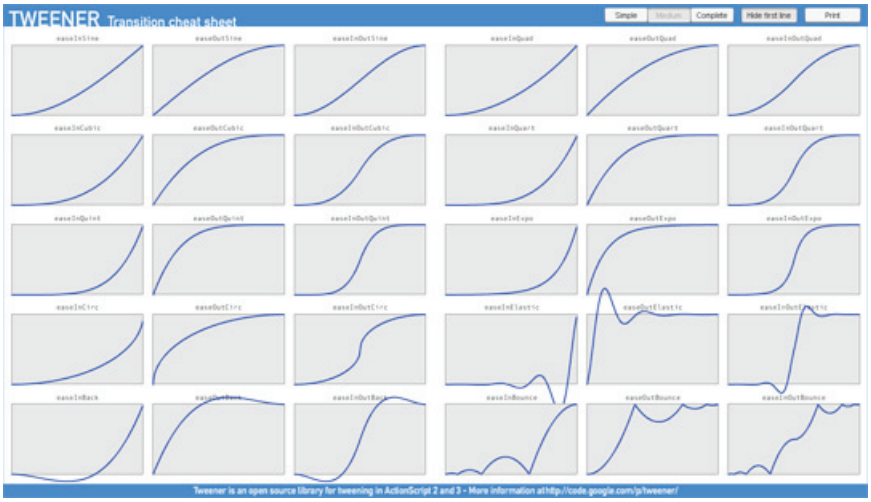
Notons que pour les mouvements complexes (p.ex. animation suivant une courbe de Bézier), il sera plus commode de réaliser celui-ci via l'éditeur de mouvement, un export vers ActionScript 3 étant alors possible via un fichier XML généré par l'éditeur.

Du côté des framework, on retiendra surtout **Tweener**, qui formalise les équations de Robert Penner.

La définition d'une animation est extrêmement simplifiée, p.ex. :

```
Tweener.addTween(target_mc, {x:0, y:0, time:1,
transition:»linear»});
```

Le type d'équation est simplement choisit, en paramètre, via la propriété transition d'un valueObject (ici «linear»). On peut juste regretter qu'il ne s'agisse pas d'un constante publique de la classe Tweener (p.ex. Tweener.TRANSITION_LINEAR).



Projet :
http://code.google.com/p/tweener/
Documentation :
http://hosted.zeh.com.br/tweener/docs/en-us/
Démo :
http://hosted.zeh.com.br/tweener/docs/en-us/misc/transitions.html

7. DESIGN PATTERN

En guise de récapitulatif, les designs patterns suivants ont été rencontrés :

Model View Controler
P.ex. pour la conception de composants graphiques sur mesure

Observable - Observer
Principalement via le modèle événementiel

Singleton
P.ex. pour la vue principale

Composite
Pourrait être appliqué pour la gestion des menus / sous-menus, mais de manière plus intéressante pour la représentation d'un système solaire (Soleil - planètes - satellites).

8. ALGORITHMES ET GRAPHISME

De manière générale, voici quelques algorithmes courrament rencontrés en graphisme.

- Algorithme de Bresenham, dessin d'un segment de droite
- Algorithme de Canny, détection des contours d'une image
- Algorithme de de Casteljau, dessin de courbes de Bézier
- Algorithme d'Oslo, dessin de B-splines
- Algorithme du peintre, masquage d'objets en 3D
- Algorithme de Sobel, détection de contours

Plus spécifiquement à ce domaine d'application, nous avons rencontré les algorithmes suivants :

L'algorithme du peintre

Celui-ci fait intervenir la notion de Z-Sorting.
L'idée vient de la peinture à l'huile: les peintres trouvaient qu'il était plus facile de peindre de fond vers l'avant-plan. Le rendering des objets par l'API 3D fait de même : la projection d'une scène 3D dans une vue 2D requiert qu'on décide quels polygones sont visibles (et lesquels ne le sont pas).

L'algorithme du peintre trie tous les polygones d'une scène par

leur profondeur, et les dessine dans cet ordre (du plus éloigné au plus proche), dessinant au-dessus des parties qui sont normalement invisibles, et solutionnant donc le problème de leur visibilité.

Ceci engendre cependant le coût de dessiner des zones redondantes (pour les objets distants). Un problème peut également survenir quand la superposition de deux objets est indécidable (même profondeur en Z et superposition).

Dans ce cas l'apport d'Andy Zupko dans l'API Away3D via le «Quadrant Render Engine» a également été porté dans l'API Papervision3D, un bel exemple d'échange d'apports respectifs entre ces deux API

L'algorithme de Perlin

Décrit dans le module 3.4.2 Animation Perlin Noise du Soleil.

5_Réflexions

1. LES FORMATS PROPRIÉTAIRES

ActionScript 3 est à la base fermé, la machine virtuelle (runtime) AVM2 l'est également.
La société Adobe, détentrice du langage, a comprit que sous le foisonnement des communautés, il fallait ouvrir le langage.

Quelques initiatives ont été prises suite à ce constat :

1) Par Adobe : gratuité du compilateur MXML (mxmlc - accessible de manière fort peu commode via un SDK), mais surtout grandes actions marketing de conquête des développeurs (via les « product evangelist» de la société, la mise à disposition gratuitement de documentation, les événements recontre, ...).

2) Par la communauté : une IDE basée sur... .Net (FlashDevelop), des framework simplifiant la conception (Gaia, ...).

Adobe a finalement laissé s'ouvrir peu à peu sa plateforme Flash. Cependant, cette décision a probablement été prise trop tard. HTML5, CSS3, jQuery, iUI, xUI, PureMVC JavaScript, ... sont les nouveaux standards qui permettent d'élaborer une solution de manière :

- **plus rapide :** le temps de développement d'un projet respectant les nouveaux standards est en moyenne 5x plus court et plus aisé à manipuler (p.ex. le debug, ...).
  Ces solutions éliminent en outre leur principal talon d'Achille (la portabilité cross-browser / plateforme), via des frameworks permettant d'abstraire ces problèmes.

- **plus légère et économe (CPU/RAM) :** prenons l'exemple de Steve Jobs qui banit le Flash Player de son iPad et prône HTML 5 (pour des raisons de consommation d'énergie).

- **plus accessible :** le référencement (SEO, ...) n'est pas un problème, alors que toute application Flash est imperméable aux

crawlers des moteurs de recherche, même si des initiatives communataires (SWFAddress) ont été prises, ce n'est qu'un pis aller.

La plateforme Flash reste toutefois un medium privilégié pour la conception d'interfaces graphiques à haut niveau de personnalisation (p.ex. Jeux, CD-/DVD-Rom). Les API 3D des solutions «standard W3C» ne sont pas encore au niveau de celles utilisées par Flash, mais cela ne saurait tarder.

Un facteur de taille reste également le nombre d'utilisateurs, garantissant un écosystème.

YouTube a migré sa solution Flash vers HTML 5... ce simple fait suffit à démontrer cette tendance.

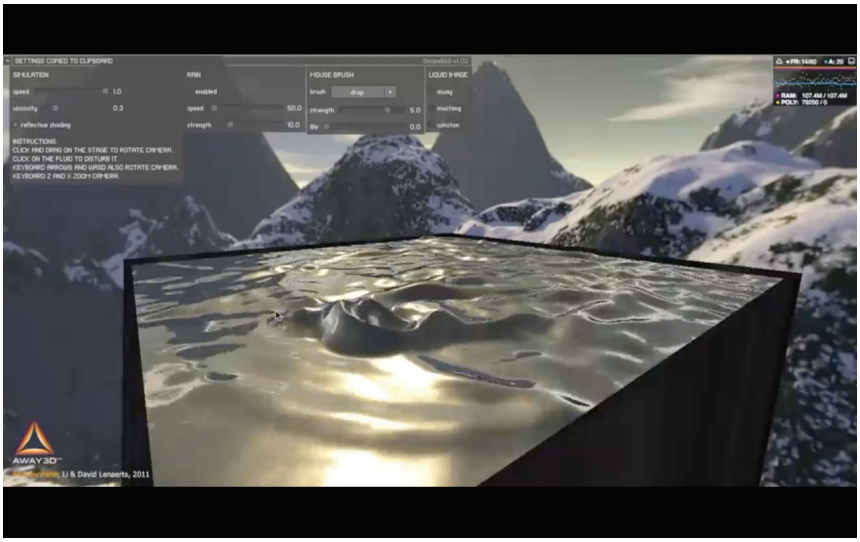
Adobe a toutefois pris l'initiative d'offrir son runtime cross platform (AIR) autant à ActionScript qu'à JavaScript / HTML.
Une position qui rappelle celle prise par Microsoft avec sa plateforme .Net par rapport à Java.

2. POURQUOI ENCORE UTILISER ACTIONSCRIPT

Si l'on souhaite être cynique, on pourrait répondre :
comme solution de fallback pour html5/css3...
C'est-à-dire pour rendre compatible des lecteurs vidéos, audio, ...
avec les anciennes versions de navigateur.

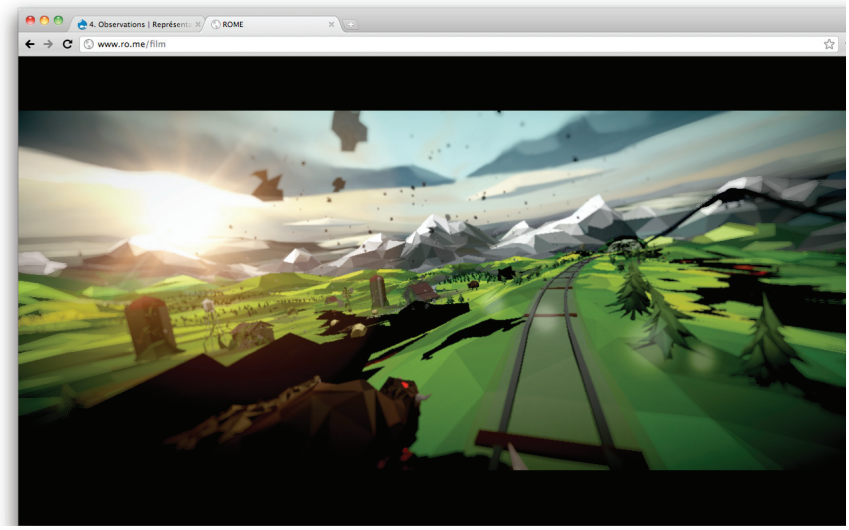
Une réponse plus pragmatique serait de lui laisser les explorations telles que la 3D haute définition, les jeux vidéos web.

Adobe collabore actuellement au développement de l'API Away3D, et les résultats sont surprenants (dans un contexte de 3D «Web») :
Molehill/Away3D 4.0 Fluid Simulation



Toutefois, les récents progrès réalisés sous Chrome en HTML 5 / WebGL et illustrés par le projet **« 3 Dreams of Black »** laisse rêveur quant aux capacités d'une communauté.

http://www.ro.me/



3. L'INTEROPÉRABILITÉ

L'enjeu actuel n'est probablement plus le hardware, le temps «machine», mais plutôt le temps «humain». En ce sens, plusieurs langages permettent actuellement de créer des ponts entre eux.

Prenons l'exemple d'Alchemy, exploitant la conversion C++ vers AS3, ce qui permets de récupérer une multitude de bibliothèques de code, notamment pour la conception de jeux (public visé par Adobe).

haXe permet p.ex. de produire un output vers JavasScript.

Notons qu'il est actuellement possible de porter vers les browser modernes une présentation pratiquement similaire à ce que nous effecturions hier avec Flash (cf. jQuery, CSS3, HTML5).

Citons notamment :

- Les appels serveur asynchrones (Ajax)
- Les interfaces riches : RIA - Rich Internet Application (jQueryUI)
- Les effets visuels : 3D, animation, ombres portées, ...

6_Sources

1. BIBLIOGRAPHIE

1. Actionscript 3 (+ API 3D)

Pratique d'ActionScript 3
Thibault Imbert
Pearson, 2010

Le meilleur d'ActionScript 3
Colin Moock
O'Reilly, 2008

ActionScript 3 Design Patterns
William Sanders & Chandima Cumaranatunge
O'Reilly - Adobe Developer Library, 2006

Learning ActionScript 3
Rich Supe, Zevan Rosser
O'Reilly - Adobe Developer Library, 2007

ActionScript 3 Cookbook
Joey Lottn Darron Schall, Keith Peters
O'Reilly - Adobe Developer Library, 2006

Professional Papervision3D
Michael Lively
Wrox, 2010

3D in Flash (Away3D)
Rob Bateman, Richard Olsson
Friends of ED, 2010

2. Esthétique computationnelle

FORM+CODE in design, art and architecture,
A guide to computational aesthetics
Casey Reas, Chandler McWilliams, LUST
Design Briefs

Getting started with processing
Casey Reas, Ben Fry
O'Reilly

3. Astronomie, Astrophysique, réflexions

Astronomie et Astrophysique
Cinq grandes idées pour explorer et comprendre l'Univers
Séguin, Villeneuve
De Boeck Université

La structure de l'univers
Alain Jorissen
Cours ULB

Cours public d'astronomie
Alain Jorissen, Benoît Famaey, Dimitri Pourbaix,
Laurent Zimmermann
CEPULB

La défaite de Platon - La science au XXe siècle
Claude Allègre
Le Livre de Poche

L'univers chiffonné
Luminet
Folio

Chimie générale
Kotz, Treichel JR
De Boeck

Portails de l'Astronomie et de la Cosmologie
Wikipedia

4. Techniques de programmation et d'analyse

La programmation Orientée Objet
Hugues Bersini
Eyrolles

UML 2 - Pratique de la modélisation
Benoît Charroux, Aomar Osmani, Yann Thierry-Mieg
Pearson

Les design patterns en Java
Les 23 modèles de conception fondamentaux
Steven John Metsker, William C. Wake
Pearson

Introduction à l'algorithmique
Cormen, Leiserson, Rivest, Stein
Dunod, 2004

Programmation
Thierry Massart
PUB, 2009

Algorithmique
Olivier Markowitch
PUB, 2009

Structures de données et algorithmes
Jean Cardinal
PUB, 2009

5. Autres technologies, formats et langages

XML in a Nutshell
Elliotte Rusty Harold, W. Scott Means
O'Reilly, 2004

Javascript La Référence
David Flanagan
O'Reilly, 2002

jQuery CookBook
Cody Lindley
O'Reilly, 2009

PHP Avancé
Eric Daspet, Cyril Pierre de Geyer
Eyrolles, 2007

HTML5 pour les Web Designers
Jeremy Keith
Eyrolles, 2011

CSS3 pour les Web Designers
Dan Cedrholm
Eyrolles, 2011

2. SOURCES D'INSPIRATION

Eléments de rédaction scientifique en informatique
Hadrien Mélot

Simulab
Framework Java développé par l'Observatoire de Paris
(http://www.obspm.fr/)

Réacteur chimique OO
Hugues Bersini

JMol
Visualisation de structures chimiques en 3D

3. CRÉDITS

Outre les API et auteurs cités dans ce rapport, une série de ressources externes ont contribué à la réalisation de ce projet.

Texture Maps des planètes et du Soleil :
http://planetpixelemporium.com/earth.html

Géolocalisation pour les autres planètes que la Terre :
http://www.geody.com/

Textes sur le soleil :
Wikipedia

Photographies :
NASA

Données relatives au positionnement des étoiles :
The Naval Observatory Merged Astrometric Dataset (NOMAD)

Je rappelerai ici les principaux contributeurs AS3 :

Créateur de Papervision3D :
Carlos Uolla

Types de données abstraits ActionScript 3 :
Michael Baczynski

Gestionnaire de layout fluide et Slideshow plein écran :
Dale Sattler

Support de l'interprétation des données du microphone :
Bartek Drozdz

Annexes

Diagrammes de classe UML

- Framework de présentation
- Voyage dans un système stellaire
- Représentation de la matière

Fichiers de configuration XML

- Application principale
- Système solaire

Sélection d'algorithmes

- Parcours récursif de la DisplayList
- Vue d'un système stellaire

Code des applications et site de démo

L'intégralité du projet se trouve en téléchargement sur le site http://ei.hypercube.be

Les modules détaillés dans la section 3.4. s'y retrouvent également sous la forme de projets distincts :

- Voyage manuel dans le système solaire
- Animation Perlin Noise du Soleil
- Composant de panneaux à scroller
- Lecteur RSS