Chrome Engine

Cahier des Charges (v2)

Propulsé par

Etudiants :

Guillaume Castellana (castel_g)
Guillaume Casalis (casali_g)
Laurent Catala (catala_l, Dublin)
Geoffroy Laptes (laptes_g, Dublin)
Dylan Oudin (oudin_d)

Table des Matières

TABLE DES MATIERES **2**

ETAT DES LIEUX **3**

FONCTIONNALITES ATTENDUES **4**

- PRODUCTIVITE 4
- PORTABILITE 4
- TEXTURES, SHADERS 4
- GESTION DES SCENES 4
- EFFETS SPECIAUX 4
- AUTRES 4

SPECIFICATIONS **5**

- TECHNIQUES 5
- APPLICATIVES 5

UTILISATION **6**

- DEVELOPPEUR 6
- UTILISATEUR 6

ARCHITECTURE PRELIMINAIRE **7**

- RACINE 7
- GESTION DE SCENES 7
- GESTION DE RESSOURCES 7
- RENDU 7

ARCHITECTURE DETAILLEE ET INTERACTIONS **8**

- LISTE EXHAUSTIVE 8

CLASSES (PREREQUIS) **9**

Etat des lieux

Le présent Cahier des Charges s'inscrit dans le projet de conception d'un moteur de rendu 3D léger et flexible permettant le développement de contenus riches sur le Web.
Le dit projet entrant dans le cadre des Epitech Innovative Projects (EIP) pour le cycle Master de l'Epitech Paris, son développement est étendu sur une période de 24 mois (Septembre 2009 – Aout 2011).

Les utilisateurs ciblés sont des développeurs habitués aux standards de codage JavaScript et ne possédant pas de connaissances approfondies en matière de programmation graphique. Cependant, le processus de création d'applications et l'architecture du moteur permettent - par le biais du polymorphisme ou de l'héritage - aux programmeurs avancés de bénéficier d'un large spectre de fonctionnalités.

Bien qu'innovant du point de vue technique, Chrome Engine se base en grande partie sur un autre moteur graphique open source : OGRE 3D.
Cette similarité s'explique par l'envie des développeurs de se baser sur un standard connu et d'offrir un mode de diffusion du contenu simple et efficace; l'univers de la programmation web se basant à 80% sur l'exploitation d'un code déjà existant.

Objectifs Juillet 2010

Productivité

- Portabilité sur les 3 principaux navigateurs du marché et les OS correspondants
- Possibilité d'instanciation de ressources (Textures, Meshs, Shaders)
- Manipulation dynamiques des ressources
- Instanciation de lumières additionnelles
- Gestion simple et intuitive des comportements Camera – Objets
- Interaction utilisateur

Fonctionnalités annexes

- Architecture du Moteur Physique terminée et développement en cours
- Gestionnaire d'entrées entièrement fonctionnel
- Fonctionnalités majeures du Gestionnaire de sons (GdS) fonctionnelles (lecture, pause, stop, gestion d'évennements)
- Fonctionnalités secondes du GdS en cours de réalisation (muxing, altération, distortion, résonance)

Documentation

- Documentation Utilisateurs complète
- Documentation Technique à jour
- Tutoriaux disponibles : First Steps, Gestion des ressources, Instanciation et manipulation
- Tutoriaux en cours de réalisation : Utilisation avancée, Ajout de modules additionnels, Développement de modules additionnels

Fonctionnalités attendues

Productivité

Utilisation en ligne / hors-ligne

Accessibilité

Rapidité

Stabilité

Documentation, support et tutoriaux

Portabilité

Systèmes d'exploitation : Windows, Mac, Linux

Navigateurs : Internet Explorer 8+, Firefox 3+, Safari, Chrome

Textures, Shaders

Rendu multi passes

Langage de programmation HLSL2 (pixel et vertex shaders)

Manipulation dynamique des textures (couleur, alpha, multi textures)

Gestion des scènes

Diversité dans la méthode de gestion (BSP, OCTree, X3)

Rendu hiérarchique

Récupération de scène par « queries »

Virtualisation des méthodes de rendu des ombres

Effets Spéciaux

Gestion des particules

Support du Billboard

Support de l'indexation et de la profondeur (Z-index ordering)

Autres

Support des formats de données communs (ZIP, TGZ, PK3)

Architecture de gestion de plug-ins

Gestion de la physique rudimentaire

Gestion des entrées / sorties

Spécifications

Techniques

Taille et format de la librairie
100Ko maximum
« G-zipped & minified »

Langage de Développement
JavaScript

Format des fichiers
.js pour la librairie et les plug-ins
.xml/.json pour les arbres et fichiers de configuration

Applicatives

Norme et convention de code
Crockford

Compatibilité avec les environnements de développement
GWT (Google Web Toolkit)
Eclipse code hints

Support des formats de texture
Textures classiques
PNG, JPEG, TGA, BMP
Textures spéciales (cubemaps, compressées, volumétriques)
DXT, S3TC

Support des formats d'archives
ZIP, TGZ, PK3

Format des rendus hiérarchiques
XML, JSON

Utilisation

Développeur

Après avoir téléchargé le Framework et inclut la librairie a l'intérieur de la page HTML, le développeur utilise l'API de la manière suivante.

```
//Création d'une Root et initialisation
myRoot = new Root()
myRoot.initialize()

//Création des gestionnaires de ressources
MeshManager = myRoot.getResourceManager("Mesh")
TextureManager = myRoot.getResourceManager("Texture")

//Chargement d'un mesh et d'une texture
MeshManager.load("totomesh", "mesh.o3dtgz")
TextureManager.load("tototexture", "texture.jpg")

//Création de Nodes pour lier le joueur et la caméra
PlayerNode = new Node("playernode", myRoot.getRootNode(), [x,y,z])
CameraNode = new Node("cameranode", PlayerNode, [x+10, y+10, z+10])

//Création d'une entité, d'une lumière ambiante et d'une caméra
Player = new Entity(MeshManager.getResource("totomesh"),
TextureManager.getResource("tototexture"), myRoot.getRootNode())
Light = new Light([x,y,z], [R,G,B], "Ambiant")
Camera = new Camera (CameraNode)

//Lancement du rendu puis shutdown (:
myRoot.startRendering()
....
myRoot.shutdown()
```

Utilisateur

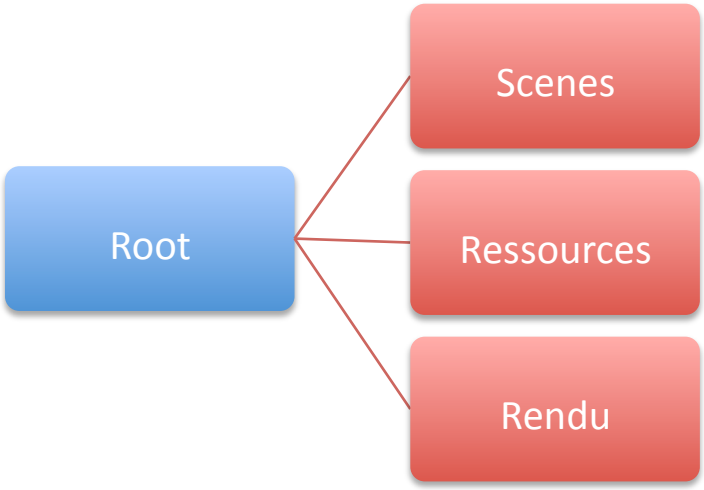
Etape 1 - Arrivée du client sur la page demandée
Etape 2 - Vérification de la disponibilité du plug-in

Etape 3A - Plug-in installé
Etape 4A - Initialisation de la fenêtre

Etape 3B - Plug-in indisponible
Etape 4B - Renvoi vers google.com/o3d

Etape 5 - Affichage

Architecture préliminaire



Racine

La racine est par définir l'entité englobant le reste des autres classes. Elle sert entre autres aux déclarations d'objets principaux tels que les Scene et Resource Managers.
Du point de vue utilisateur, c'est le point de départ du projet ; un organiseur qui permet de créer les outils pour gérer le tout en détail par la suite.

Gestion de scènes

Cette partie gère l'architecture et la gestion de la scène, la façon dont elle est structurée et la manière dont elle est perçue par la/les caméras.
Elle sert majoritairement d'espace de configuration et de déclaration pour les différentes entités du projet.

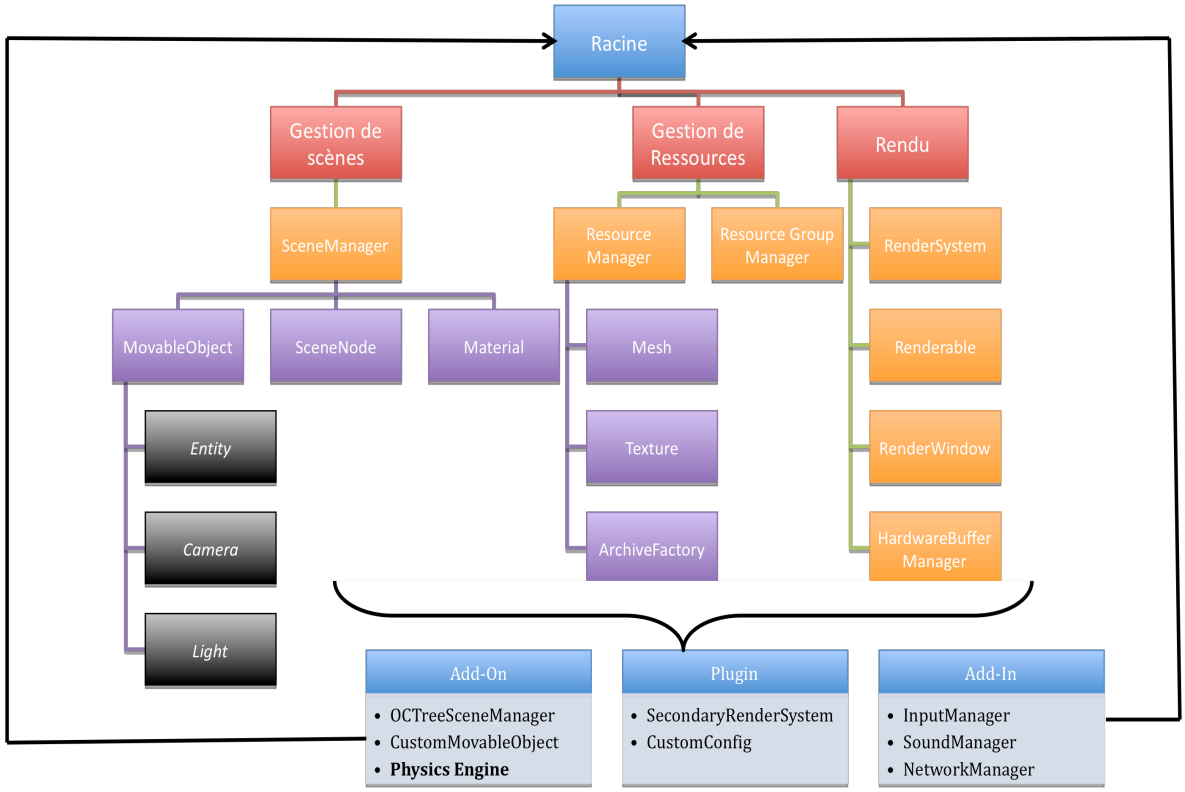
Gestion de ressources

Cette partie s'occupe des ressources : le chargement de textures ou d'archives, leur décompression, leur déchargement et leur gestion en mémoire.

Rendu

Le rendu affiche les entités du projet à l'écran, il s'agit de programmation bas niveau gérant le pipeline graphique, le buffer matériel et les états logiques de rendu.
Les composants du gestionnaire de scènes tirent leur fonctions haut niveau de ce composant.

Architecture détaillée et interactions



Liste exhaustive

Resource
- Font
- Texture
- Material
- Mesh

MovableObject
- Entity
- Camera
- Light
- ParticleSystem
- Billboard

Node

ResourceManager
- MeshManager
- TextureManager
- LightManager
- CameraManager
- ParticleManager
- BillboardManager
- ArchiveFactory

FrameListener

Renderer

Root

Singleton

Classes (prérequis)

Root

- GETTERS
- SETTERS
- initialise()
- startRendering()
- shutdown()
- toggleFullscreen()
- addCamera()
- removeCamera()

- ***mVersion***
- ***mConfigFileName***
- ***mResourceManagers***
- ***mRenderer***
- ***mFrameListeners***
- ***mRootNode***
- ***mCameras***
- ***mCursor***
- ***mHeight***
- ***mWidth***
- ***mClientID***
- ***mClientInfo***
- ***mFullScreen***
- ***mInitialised***

Resource

- prepare()
- load()
- reload()
- unload()
- isLoaded()
- isLoading()
- addListener()
- removeListener()
- GETTERS

- ***mCreator***
- ***mName***
- ***mGroup***
- ***mHandle***
- ***mLoadingState***
- ***mSize***
- ***mOrigin***
- ***mType***
- ***mListenerList***

MovableObject

- GETTERS
- SETTERS

- ***mCreator***
- ***mName***
- ***mParentNode***
- ***mVisible***
- ***mListener***
- ***mLightList***

Node

- GETTERS
- SETTERS
- scale([x,y,z])
- translate([x,y,z])
- rotate([x,y,z])

- ***mName***
- ***mParent***
- ***mChildren***
- ***mPosition***
- ***mOrientation***
- ***mScale***
- ***mWorldMatrix***
- ***mLocalMatrix***
- ***mListener***

ResourceManager	FrameListener	Renderer
• GETTERS • SETTERS • unload() • unloadAll() • reload() • reloadAll() • remove() • removeAll() • getByName() • getByHandle() • resourceExist() • getResourceType() • ***mMemoryBudget*** • ***mMemoryUsage*** • ***mResources*** • ***mLoadOrder*** • ***mResourceType***	• Callback() • ***mName*** • ***mHandle*** • ***mExecuteOrder***	• GETTERS • SETTERS • ToggleFrameRate() • AddFrameListener() • RemoveFrameListener() • ***mRenderMode*** • ***mCulling*** • ***mFrameRateVisibility***