

Interview de Jerome [JeGX] Guinot

Réalisé par Christophe [Groove] Riccio

27/05/2006

Présentation

JeGX est un programmeur amateur de longue date qui excelle dans le rendu temps réel. Depuis quelques années, il développe un outil nommé [Hyperion](#) dont les premières versions sont disponibles depuis 2005. JeGX a accepté de répondre à cette interview dans lequel et nous explique son parcours, son aventure et ses projets futurs.

Groove:

Quel est ton parcours amateur, scolaire et professionnel?

JeGX:

D'aussi loin que je me souviens, j'ai fait des études d'électronique. A cette époque obscure, où le 386 n'était pas encore la puce révolutionnaire, j'ai dû toucher une ou deux fois à un clavier mais sans plus. J'ai véritablement découvert l'univers de la programmation en licence. Coup de pot! J'ai choisi une licence qui mixait 3 matières que j'affectionnais à l'époque: électronique, mathématique et physique (licence EEA à Paris 6 si ça existe toujours). Mais cette licence avait en plus une quatrième matière: l'informatique.

Et j'avoue que le fait de me retrouver plongé directement dans des cours d'assembleur 68000 et de programmation C m'a filé une bonne claque. Je n'avais jamais écrit une ligne de code et me voilà en pleine séance de Travaux Dirigés où je devais comprendre le maniement de la pile en ASM 68000... gloup! Mais le déclencheur de ma passion pour le code a été le moment où j'ai pu mettre la main sur un compilateur C: le turbo c++ de Borland livré sur 2 disquettes 1.44M (trop fort!). Au même moment, j'ai réussi à monter tant bien que mal ma première vraie bécane basée sur un 386 DX 40 de AMD (négocié chez un asiatique rue Philippe Auguste en face de là où il y avait anciennement Surcouf...) avec 4M de ram!

Ce qui m'a vraiment mis sur les rails du code a été l'acquisition du book "La Bible PC" 5ème édition. Un pur livre 200% programmation système C / asm x86. J'ai vraiment appris à coder le C avec ce book. J'ai passé des jours et des nuits à coder des routines systèmes, des infecteurs de COM et d'EXE et tout ce qui pouvait se coder sur le pc. A côté, les cours de la fac ne valaient pas grand chose. Je crois que c'est en 96 que j'ai découvert ma passion pour la programmation graphique avec 3 choses: le Magazine PC team qui publiait chaque mois un article sur la démo scène (dommage qu'ils aient arrêté, c'était vraiment bien et à l'époque le magazine était réellement instructif avec ses rubriques programmation) et la sortie en librairie de plusieurs livres vraiment cool: le zen de la programmation graphique d'Abrash [1], PC interdit et Programmation de démos en C. J'entrais dans l'ère du mode 13h [2] et de mes premiers polyfillers C/ASM. Depuis ce moment je ne code que du GFX.

Mon passage à l'armée m'a lobotomisé les neurones et à la sortie je me suis retrouvé chez une grosse compagnie américaine (dont je tairais le nom - pas de pub gratos!) à dépanner des pc! Puis je me suis retrouvé dans une petite boite en tant que business développeur (il y a deux types de développeurs: les Business Dev et les Game Dev - Ce n'est pas méchant, c'est juste pour différentier) et DBA [*NDLR : DataBase Administrator*] Oracle (bonjour, moi qui n'avait jamais touché à une Base de donnée ou à une requête SQL). En 2001, je me suis retrouvé dans une Start-up à développer des routines graphiques 2D pour de la cartographie de données et après dans une boite Suisse à développer du code pour un système de particules 3d.

A cause des impératifs professionnels de ma femme, nous devions sans cesse bouger à droite et à gauche en Suisse, du coup avec les petites réserves que j'ai pu mettre de coté, je me suis mis tranquillement dans le développement d'un moteur 3d qui à abouti aujourd'hui à [Hyperion](#) et à ma fin prochaine ! (*NDLR : Comprenez, c'est du boulot a en crever, pas de 35H dans la profession ;*)

Notes de la rédaction

[1] Micheal Abrash est un très célèbre programmeur reconnu pour sa passion de l'optimisation extrême. Il est également connu pour son lien avec John Carmack dont il était le mentor lors de son arrivé sur le PC. Il a écrit de nombreux livres sur l'optimisation qui sont à présent compilés dans le [Michael Abrash's Graphics Programming Black Book Special](#) ou en version Française, [Programmation Graphique C/C++ Assembleur](#). Bien que basé sur de vieux processeurs du 8086 au Pentium, c'est toujours une référence car il est au delà de la programmation, c'est un livre philosophique pour le programmeur graphique.

[2] Le mode 13h est l'appellation usuelle de la fonction 13 de l'interruption 21 qui permettait d'utiliser le mode graphique le plus adapté au rendu graphique temps réel du DOS : Une résolution en 320*240 avec 256 couleurs, le pied à l'époque.

Groove:

[Hyperion](#) est un outil définitivement original avec une approche très moderne. Peux-tu décrire cette approche, la raison de ce choix et ton but à terme?

JeGX:

[Hyperion](#) est né il y a environ 2 ans lorsqu'un pote, qui avait une Start-up dans les arts graphiques et pour qui je développais des petites démos 3D, avait envie de plus d'autonomie pour la réalisation de ces démos. Les démos étaient toujours plus ou moins les mêmes: modèles 3D, lumières, skybox, les objets classiques pour une scène 3D de base. Il m'avait demandé si ce n'était pas possible d'avoir un fichier de configuration pour la démo dans laquelle il suffisait de changer le filename du modèle, la position et la couleur des lumières... Sans le savoir il a lancé les bases d'[Hyperion](#).

Très rapidement la première version d'[Hyperion](#) à vu le jour en juin 2004 (ça commence à dater tout ça...). Le script avait une syntaxe propriétaire, ressemblant un peu à la

description d'un fichier ASE [1] de 3DStudio. Cependant, plus le nombre de fonctionnalités augmentait et plus le parser devenait lourd. De plus, visuellement c'était assez repoussant mais au détour d'une net surf session, je suis tombé sur un article sur la [Libxml](#) [2]. A partir de ce moment là c'était clair: toute la description statique de la scène se ferait en XML. C'était d'une évidence, on se demande même pourquoi je n'avais pas commencé par ça...

Dans ma tête, [Hyperion](#) avait deux objectifs: d'une part, il devait être un outil qui me permette de coder des scènes 3D sans prise de tête, sans compilateur C++, avec une syntaxe très simple d'accès et quasi naturelle, d'autre part, il devait me permettre de faire des choses plus complexes comme exploiter les shaders temps réel (les shaders OpenGL pour être plus précis), accéder aux vertices des meshes, récupérer les buffers audio pour de la synchro musiques/GFX ou d'autres trucs tordus.

Pour gérer tout le coté dynamique et toute la logique de la scène 3d / démo, il fallait un langage de scripting. Un langage simple, très simple de préférence et gratuit. J'ai trouvé [LUA](#) [3] à ce moment là. J'en avais vaguement entendu parlé mais sans plus. J'ai commencé à le tester et quelques heures après il était opérationnel dans [Hyperion](#)! J'ai flashé dessus. Pour le développeur, [LUA](#) est d'une simplicité enfantine à intégrer dans son application et pour le end-user [LUA](#) est encore plus simple à utiliser, mais qu'on ne s'y trompe pas, sous cette simplicité apparente, [LUA](#) est un langage extrêmement souple et puissant. Je n'arrête pas de découvrir des nouvelles façons de l'utiliser. En gros que du bonheur! Avec le XML et le [LUA](#), [Hyperion](#) avait presque tous les ingrédients pour que prenne la sauce.

Il en manquait un et de taille: la possibilité d'intégrer des shaders temps réel. On ne peut plus envisager aujourd'hui de coder une démo sans shaders. Ils apportent une réelle souplesse dans la mise au point d'effets graphiques. Prenons par exemple le bump mapping. Avant les shaders, il fallait se plonger dans les textures combiners, ajouter une cube map de normalisation [4] et encore on obtenait difficilement des reflets spéculaires dignes de ce nom. Avec les shaders, la texture de base et la normal map sont suffisantes et quelques petites lignes de code en GLSL permettent d'aboutir à un bump mapping plus vrai que nature. Tout ça pour dire que les shaders GLSL (oui, [Hyperion](#) repose sur un plugin OpenGL) font partie intégrante d'[Hyperion](#) et leur mise en oeuvre est d'une simplicité extrême puisqu'il suffit de créer un noeud XML pour le shader, d'y loger le code GLSL et de spécifier la cible du shader.

Maintenant, j'essaie de rendre la Host API [Hyperion/LUA](#) (nom donné à l'interface de programmation entre [LUA](#) et [Hyperion](#)) la plus polyvalente possible si bien qu'à terme une démo complète [Hyperion](#) pourra être codée qu'avec du [LUA](#). Je dois avouer, surtout avec la dernière version d'[Hyperion](#), que de coder une scène 3D en C++/OpenGL me semble presque d'un autre monde, un truc que faisaient les anciens. Je ne me replonge dans les entrailles du moteur 3D que lorsque c'est nécessaire, pour ajouter une nouvelle fonctionnalité ou pour corriger un bug mais à chaque fois c'est la même impression : c'est dépassée.

De plus, je voulais qu'[Hyperion](#) puisse être un outil didactique pour s'initier au monde de la programmation 3d temps réel. C'est pour cela que tout le vocabulaire qui sert à la construction et l'animation d'une scène 3d est très commun. Pourquoi réinventer la roue? Un exemple simple: le blending. En OpenGL ou en Direct3D, il y a la notion de facteur source et destination avec des valeurs du style D3DBLEND_SRCALPHA pour d3d ou GL_SRC_ALPHA pour ogl. Avec [Hyperion](#), le même facteur de blending s'écrit BLENDING_FACTOR_SRC_ALPHA. De même, les VBOs, les vertex buffers objects d'OpenGL, peuvent être activés ou non pour le rendu d'un mesh, et les différentes politiques de gestions des VBOs peuvent être contrôlés (STATIC, DYNAMIC ou STREAM) [5]. On voit donc qu'au niveau des concepts et notions existantes dans les API 3D modernes, Hyperion se révèle aussi compliqué à utiliser qu'OpenGL ou Direct3D. Du coup, un book sur ogl ou d3d peut servir pour vraiment comprendre la signification de tel ou tel paramètre.

J'en ai même profité, à l'occasion d'une démo sur le rendu des fractales de Mandelbrot, pour ajouter les fonctionnalités nécessaires au GPGPU (General Programming on GPU): les rendu en ping-pong [6] dans les textures en virgule flottante. Pour y arriver, le end-user hyper ionique peut manipuler directement les FBOs (bind, color_attachment, etc.) [7]. Fort heureusement, [Hyperion](#) possède des valeurs par défaut pour tous les paramètres et les plus complexes restent masqués pour celui qui ne veut pas en entendre parler!

Un des autres objectifs d'[Hyperion](#) est de pouvoir être un jour utilisé dans la démo scène. J'affectionne particulièrement cet univers mais je sais, en France en tout cas, que la démo scène est en chute libre. Peut être qu'un système comme [Hyperion](#) pourra offrir à de jeunes démo makers en herbe une plateforme facile d'accès pour mettre au point rapidement des productions pour des démo parties. Je le souhaite en tout cas. Si un jour, des démos makers veulent utiliser [Hyperion](#) pour une production, qu'ils me le fassent savoir et je n'hésiterais pas à ajouter ce qu'il faut à [Hyperion](#) pour qu'ils bouclent leur démo.

Tout ce baratin c'est bien beau mais est-ce que ça fait vivre? Pas encore mais doucement les premières demandes d'acquisition d'une licence commerciale, par des professionnels, arrivent et j'espère pouvoir, un des ces quatre, monter une boîte pour développer sérieusement Hyperion et lui donner enfin toutes ses chances. Pour l'instant, je suis seul et ça commence à être quelque peu épuisant ! Quoi qu'il en soit, pour une utilisation non commerciale, [Hyperion](#) est ET restera 100% gratos et 100% fonctionnel.

Notes de la rédaction

[1] ASE est un format de description de maillages non animée dans un fichier texte

[2] [Libxml](#) est un parseur XML notamment utilisé par l'interface graphique [Gnome](#) sous Linux. Elle supporte les méthodes SAX et DOM pour traitement des données et permet la validation de fichier XML à l'aide de DTDs

[3] LUA est un langage de script très utilisé par les jeux vidéo actuels.

[4] Les texture combiners sont une spécificité d'OpenGL 1.3, décrite par l'extension [GL_ARB_texture_env_combine](#). C'est la partie de l'API qui permet de manipulé les textures les unes avec les autres, les fonctionnalités sont globalement équivalent au pixel shader 1.* de Direct3D 8 mais d'une lourdeur jamais égalé.

[5] Le VBO est une nouveauté d'OpenGL 1.5 avec l'extension [GL_ARB_vertex_buffer_object](#) qui permet notamment de « choisir » l'emplacement de stockage des données des meshes.

[6] Rendu ping pong est l'expression utilisé pour désigner le swap permettant entre les textures utilisés pour le rendu sur texture en [GPGPU](#).

[7] Le Frame Buffer Object (FBO) est l'extension [GL_EXT_framebuffer_object](#) d'OpenGL permettent de rendre directement une scène dans un texture.

Groove:

Selon tes propos "[Hyperion](#) repose sur un plugin OpenGL", qu'est ce que ce plugin? La notion de plugin connote une externalisation du code du renderer, quel est le but de ce choix ? Envisages-tu une version pour Direct3D et HLSL?

JeGX:

Effectivement, mon cher Groove, [Hyperion](#) repose sur un plugin OpenGL. Il serait plus juste de dire: [Hyperion](#) repose sur le moteur [oZone3D](#), ce dernier étant livré avec un plugin OpenGL. Il s'agit d'un vrai plugin ce qui se traduit comme tu le dis par une externalisation du code du renderer.

Le moteur oZone3D est composé d'un noyau (le core) et d'une série d'interfaces qui permettent de développer des plugins pour tout ce qui est externalisable: loaders 2D/3D, sound system, network, renderer 3D temps réel, particles system. D'ailleurs en jetant un coup d'oeil à l'arborescence d'[Hyperion](#), on peut voir ces différents modules [1] :

- le kernel (oZone3D_Main.dll)
- les loaders 3D (o3_plugin_data_loader_3d_3DS.o3p, etc.)
- le sound système (o3_plugin_sound_system_fmod.o3p)
- le renderer OpenGL (oZone3D_Rendering_Core_OpenGL.o3p)
- ...

Cette architecture est souple et permet de limiter les dépendances entres les basses couches (les plugins) et le kernel du moteur. Finalement, le kernel ne contient que le code nécessaire à son fonctionnement et n'est en aucune façon lié au code des plugins si ce n'est par l'interface des plugins. Ce qui est cool aussi avec cette architecture, c'est qu'une fois que tu as développé ton plugin, à condition que ce que tu as codé soit correct, le moteur saura l'utiliser. Dans le cadre d'une entreprise, cette architecture prend toute sa valeur: l'entreprise n'est pas obligé d'embaucher des développeurs connaissant la 3D pour développer des plugins loaders 2D, sound system ou réseau qui seront pourtant utilisés par le moteur 3D. Plusieurs spécialités pourront ainsi travailler de concert, sans même

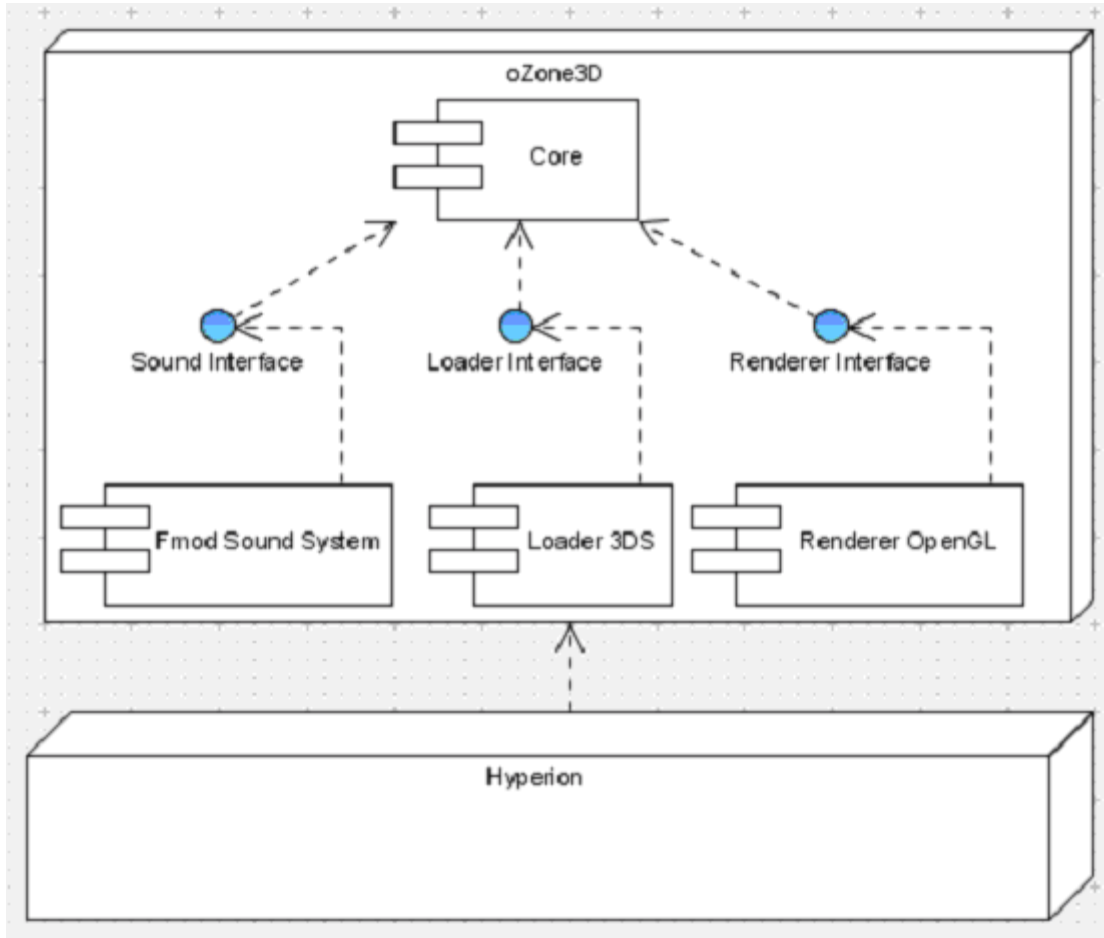
jamais se voir ni se parler, à la fabrication du moteur 3D complet. Offshore quand tu nous tiens!

Dans la terminologie [oZone3D](#), l'interface du renderer temps réel est appelée Rendering Core. Cette interface définit environ une centaine de méthodes (enfin je crois, j'ai jamais compté) purement abstraites qui ne demandent qu'à recevoir un corps: cette demande est satisfaite en implémentant le plugin. Au tout début du moteur, je développais les 2 plugins OpenGL et Direct3D en parallèle mais pour d'obscures raisons, qui aujourd'hui m'échappent totalement, j'ai laissé tomber le développement du plugin Direct3D pour me concentrer sur celui OpenGL mais j'imagine que j'ai du laisser tomber le plugin Direct3D pour des raisons de temps.

De plus, j'affectionne particulièrement OpenGL pour son API qui est simple car ce n'est en gros qu'une longue liste de fonctions C. J'adore ce concept, à savoir une librairie de base en C, car elle te laisse libre de créer toi même tes propres classes et de modéliser ton moteur 3D et ton application en fonction de ta façon de percevoir comment doit fonctionner un système. Finalement, utiliser des classes et objets C++ c'est, en quelque sorte, utiliser la façon de raisonner et de penser du développeur de ces classes. Mais bon ne nous égarons pas.

Si un plugin D3D voit le jour, automatiquement le support du HLSL sera intégré puisque c'est son langage de shading par excellence. Le re-développement d'un plug d3d est une chose que j'ai en tête depuis un certain temps mais j'attends de voir ce qui se passe avec DX10 car, bien que je n'aie pas trop regardé, j'ai lu qu'il y a des changements au niveau de l'interface entre DX9 et DX10. Mais une chose est sûre, le jour où j'ai les ressources nécessaires disponibles, je lance le développement du plug d3d.

Notes de la rédaction



*Représentation sommaire de l'architecture d'Hyperion
par un diagramme de déploiement UML*

Groove:

Tu nous informes qu'"[Hyperion](#) se révèle aussi complexe à utiliser qu'OpenGL ou Direct3D". Certes avec un livre OpenGL ou Direct3D le développeur pourra se renseigner sur les fondements de la 3D temps réel, mais le langage XML reste propre à [Hyperion](#). Comment communique-tu dessus?

JeGX:

Un des objectifs d'[Hyperion](#) est de permettre, à toute personne n'ayant pas de notions particulières en 3D, d'aborder le domaine du développement de petites applications 3D temps réel avec facilité. A chaque fois que je le peux, j'essaie de coller avec les termes courants employés par les API Direct3D et OpenGL de façon à ce que les notions et concepts utilisés et compris soient exploitables en dehors d'[Hyperion](#). Je ne veux absolument pas enfermer quelqu'un dans [Hyperion](#).

Au niveau XML, beaucoup d'attributs prennent des valeurs connues en OpenGL (facteurs de blending, facteurs de comparaison alpha testing, mode de filtrage et d'adressage des

textures, coefficients des matériaux) et au niveau host-API [Hyperion/LUA](#) c'est encore plus visible: la fonction de la bibliothèque camera qui permet d'agir sur la matrice de vue s'appelle SetViewParams. Cela ressemble étrangement à Direct3D, non? Au niveau de la librairie Lighting, les fonctions portent des noms tels que SetSpotCutOff, SetQuadraticAttFactor ou SetDiffuse: là on copie dangereusement OpenGL.

La programmation 3D est un domaine extrêmement fun et intéressant mais aussi extrêmement vaste, complexe et exigeant. Un débutant pourrait parfois arrêter et abandonner avant même d'y avoir vraiment goûté. Quel gâchis. Avec [Hyperion](#), j'espère pouvoir faire découvrir aux débutants ce monde passionnant et lui inoculer le virus afin qu'ensuite s'il le souhaite, il puisse se plonger de lui même dans la programmation hardcore c/c++/opengl/direct3d. Lorsqu'il sera confronté à ces API, il sera en terrain connu sans même s'en apercevoir.

Groove:

En certains aspects, [Hyperion](#) me rappelle [FX Composer](#) et [RenderMonkey](#). Que penses tu de ces softs, sont t'ils une source d'inspiration? Souhaites-tu t'orienter vers une application de ce type ?

JeGX:

RenderMonkey (RM) et FX Composer (FXC) sont des environnements principalement conçus pour la mise au point des shaders. Tu peux difficilement contrôler comme tu le veux ta caméra, ou charger et effectuer des traitements sur tes modèles 3D, mettre du son ou transmettre des données via le réseau. Dans RM, par exemple, les seules zones de codes éditables sont celles des shaders (enfin je crois, mais j'ai rien vu d'autre).

[Hyperion](#) vise à être un environnement généraliste pour la mise en place d'une scène 3D interactive ou non (pure démo). Le développement d'un jeu complet est possible même si personne ne l'a encore fait. Dans le concept de base, RM, FXC et [Hyperion](#) sont de même nature: permettre à des développeurs ou non de mettre en place ou d'expérimenter rapidement un effet. De plus, derrière l'interface graphique de RM ou de FXC, se cache un gros fichier XML qui décrit la démo. C'est le même principe qu'[Hyperion](#). Pour l'instant, j'essai de puiser des idées de démos à partir des projets d'exemples livrés avec RM ou FXC afin de donner à manger à [Hyperion](#) pour le faire grandir. La démo [water réflexion](#) par exemple.

Je pense que je me concentrerai sur la façon dont l'interface de RM ou de FXC est organisée lorsque le développement d'une GUI pour [Hyperion](#) sera à l'ordre du jour. Une GUI (graphical user interface) permet un gain de vitesse pour les opérations classiques, c'est indéniable, mais j'attends d'avoir plus de ressources pour m'y attaquer car une GUI digne de ce nom est un sacré boulot!

D'un autre coté, l'absence de GUI et une structure du code simple et clair permet, avec un tout petit peu d'habitude de développer ses scènes 3D relativement vite. De plus, une GUI ajoute des problèmes pour la portabilité de l'application. Un des objectifs à moyen terme est de porter le moteur 3D et [Hyperion](#) sous Linux. Le code du moteur ne devrait pas

poser trop de problèmes et une interface aussi simple que l'interface actuelle d'[Hyperion](#) non plus. Mais je pense qu'à terme, les deux façons de produire du code seront disponibles (GUI et/ou codage hardcore).

Pour terminer et pour être tout à fait honnête, une GUI est quelque chose de relativement lourd et qui au fond n'apporte rien à la productivité lorsque tu commences à maîtriser le code Hyperion (ce qui, au passage est relativement rapide). Donc je pense que seule une très forte demande de la communauté des utilisateurs [Hyperion](#) pourra lancer le développement d'une GUI...

Groove:

Qu'entends-tu par manque de ressources? Du temps, de l'argent, des connaissances, des bras? As-tu des projets, des idées pour combler ce manque?

JeGX:

Le manque de ressources est clairement un manque de bras et d'argent, mais je pense que c'est le cas de la plupart des projets de ce type. Je suis pour l'instant tout seul pour le développement du moteur 3d, des [softs](#) ([Hyperion](#) principalement), la maintenance du [site](#) et la rédaction des [tutoriaux](#) que j'essaie de sortir toutes les 3 semaines (il y a des périodes où ça le fait et d'autres, comme en ce moment, où je suis total à la traîne!). S'il y a des personnes intéressées par la rédaction de tutoriaux qu'ils me contactent (jegx_AT_ozone3d_DOT_net), je serais heureux de leur laisser la possibilité de s'exprimer au travers d'un tutoriel sur un sujet précis du graphisme 3D sur ordinateur (tous sujets confondus: techniques de modélisation, programmation 3D, autres...).

Concernant des sources de financement, je n'ai pas encore d'idées très précises et j'y travaille mais ce n'est vraiment pas évident. Je suis du genre à avoir pleins d'idées et à rester à les concrétiser derrière mon ordi plutôt que faire grandir le business. [Hyperion](#) est une des sources car j'ai reçu quelques demandes fermes pour acquérir une licence commerciale dont une qui vient de se concrétiser. Si quelqu'un possède la fibre commerciale pour vendre ce genre de soft, qu'il me contacte, je pense qu'il y a un marché à explorer et sûrement à exploiter. Je vais commencer tranquillement à propager la news sur certains sites et aussi contacter quelques magasins.

Le coté pub sur le site aussi peut être une petite source de revenu. Les tutoriaux sont un excellent moyen de faire connaître le site et d'attirer à terme des annonceurs. C'est pour cela que j'y consacre une bonne partie de mon temps. J'espère que ce travail sera payant dans les prochains mois.

En attendant, je viens de concrétiser un petit contrat avec une université suisse qui m'apportera un peu d'oxygène tout en me permettant de me concentrer sur [oZone3D](#) et ses dérivés... Localement (sur [Genève](#)), il y aurait une source de revenue avec des formations au niveau 3d temps réel. Cela va être mis en place très prochainement et je pourrais peut être avoir besoin de "profs" pour ces formations. Je posterai une news sur [Game-Lab](#) à ce sujet.

Comme on dit: "la perseverenza è la madre del successo" alors je continue. C'est vraiment hardcore de réussir à lancer un projet mais la liberté n'a pas de prix et lorsqu'on y a goûté, il est impossible de revenir en arrière. Affaire à suivre !

Conclusion

C'est avec un grand plaisir que j'ai réalisé cette interview de JeGX. J'en profite pour le remercier pour sa coopération complète, pour la richesse de ses explications et enfin pour les espoirs qu'il apporte à la communauté amateur. Il prouve qu'avec l'envie, un amateur peut aller loin et peut-être toucher ses rêves. Je souhaite à JeGX d'atteindre son objectif, vivre de sa passion sans concession.