Cahier des Charges

2B2S

18 novembre 2005

Thomas 'billich' De Grivel
Maxime 'loucha_m' Louchart
Bruno 'Broen' Malaquin
Julien 'Splin' Valentin

Patchwork13!

Table des matières

1 Introduction

Patchwork13 est notre projet pour cette année de Spé à l'EPITA.

C'est un outil de synthèse modulaire, capable de générer tous types de données et de leur faire subir tout type de traitements supposants une évolution dans le temps. Les données sont crées et modifiées par des modules (que nous appellons des *patches*) grâce à un système de plug-ins. Ces patches sont reliés entre eux pour former un graphe, éditable par le biais d'une interface graphique. Les données transitent alors de patch en patch à chaque instant T du calcul.

Nous permettons ainsi à l'utilisateur lambda (non programmeur) de realiser **à la souris** un nombre illimité de fonctions complexes qui, grâce à l'aspect modulaire, deviennent très faciles a mettre en œuvre. Il faut bien sûr pour cela que les modules nécessaire à l'application aient été ecrits auparavant, mais une fois écrits, plus besoin de savoir programmer pour les utiliser.

Pour donner quelques exemples restreints, les possibilités vont du simple calcul entier à la synthèse audio (que nous traiterons), aux réseaux de neurones en passant par des effets videos que l'on pourrait chaîner á l'infini, ainsi que la génération d'animations complexes en trois dimentions (que nous traiterons aussi).

Pour en savoir plus, merci infiniment, par avance, d'avoir l'amabilite de bien vouloir – en ayant reçu toute l'assurance ainsi que la gracieuse expression de nos sentiments les plus distingués – tourner cette page.

2 Nature du projet

Le projet sera écrit en C standard pour maximiser la portabilité, et sera notament porté sous Mac OS X et Windows. Il sera composé d'une interface graphique (pw13_gtk) utilisant une librairie "noyau" (pw13) elle-même capable de charger diverses librairies de patches.

Ce projet est inédit, du moins à nos yeux. Nous l'avons défini par le terme 'outil de synthese modulaire' (modular synthesis toolkit). Certains diront 'boîte à outils' pour souligner le côté conceptuel de la réunion de nombreuses fonctionalités. Pourquoi pas mais au final nous voulons créer un logiciel permettant la composition de modules et bien plus encore. Le projet se présentera par le biais d'une interface graphique permettant le choix et la connexion des modules, et donc l'édition du graphe du processus. Ce processus pourra à l'extrême être tout et n'importe quoi.

Ce caractère illimité du projet lui apporte une grande richesse du côté de l'utilisation et des possibilitées. Générez une, deux, trois valeur(s) et appliquez leur quelques dizaines de traitements bien placés et voyez le résultat des calculs se faire en temps réel !

De plus un accent sera mis sur la simplicité : si tout les modules sont présent pour faire ce que l'on desire, en quelques clics le projet voulu sera élaborét si l'on veut plus, pas de de problême puisque la conception de patch sera facilité par une structure simple pour ne pas décourager les éventuels développeurs. Finalement c'est un logiciel aidant à la conception d'outils en rassemblant et en mettant à disposition l'orchestration de plus petits marteaux.

3 Origine du projet

Le projet Patchwork13 ! s'inspire ouvertement des debuts de la musique electronique et de ses premiers synthetiseurs.

Ceux-ci etaient dits modulaires car faits d'un assemblages de modules appelés patches (comme dans patchwork). Chaque patch effectuait un traitement simple sur le son, et une fois connectés les uns aux autres permettaient d'effectuer un traitement complexe.

Cela offrait evidemment une liberte de composition des patches virtuellement infinie, et surtout une abstraction du fonctionnement de chaque patch : nul besoin d'etre electronicien ou de savoir faire un patch pour les connecter, il suffisait de brancher un cable electrique.

De nombreux logiciels de musique ont deja repris ce principe et permettent de dessiner un graphe de patches, mais ils ont tous un inconvenient commun : ils ne permettent de manipuler que du son (des scalaires).

D'autre part il existe très peu de logiciels permettant une réelle interaction entre les modules (controle de l'un par l'autre), puisqu'on ne peut generalement que connecter la sortie son d'un module à l'entrée son d'un autre. Autrement dit on ne peut pas connecter les paramètres de ces modules. (Par exemple si on considère un module "filtre", on peut lui connecter en entrée le signal a traiter, mais pas sa frequence propre : elle reste un parametre à rentrer “à la main”).

Le projet Patchwork13 reprend litterallement le concept de patch, et l'étend largement en permettant de typer les données que les patches manipulent.

Les applications possibles depassent alors largement la musique puisque la definition de nouveaux types de données est possible.

Un patch, peut alors être assimilé à une fonction à n paramètres typés, renvoyant p valeurs typées. Cela n'est pas sans rappeler la programmation fonctionnelle, à laquelle on aurait ajoute une gestion du temps. Nuance non négligeable puisque les languages fonctionnels actuels utilisent tous un *garbage collector* qui peut induire des temps de latence aléatoires, totallement incompatibles avec une application temps réel.

4 Intéret du projet

Le but principal de ce projet est d'apporter la possibilité à des personnes qui n'ont pas forcément beaucoup de connaissances en informatique, de pouvoir "programmer" visuellement. En effet, le système de patchs gère l'enchaînement de patchs les uns a la suite des autres. Cela permet, pour un exemple basique, d'utiliser un patch qui va faire une sorte de `scanf` puis faire une liste d'opérations mathématiques grâce à d'autres patchs et d'en afficher la sortie avec un patch `printf`.

L'intérêt de ce projet est qu'il est un vrai "couteau suisse" permettant d'obtenir relativement simplement une combinaison d'opérations plus ou moin complexe. De plus l'integration d'un clustering a pour but de rendre cet outil utilisable pour des taches qui requiérent enormément de ressources système, accélérant donc ainsi le temps d'éxécution.

Sans vouloir paraître vantard, ce projet possède l'avantage d'avoir des possibilités infinies. Personnellement nous ne développerons pour le moment essentiellement que des libraires son, video et les libraires standard accompagnées de patchs respectifs. Cependant il faut avoir à l'esprit que le projet n'a pas de bornes limitant son intérêt : il est tout a fait possible d'intégrer toutes les fonction d'un logiciel de création sonore comme frutty loops, d'avoir des patch permettant de générer du pdf avec un simple texte ou encore de greffer un module servant à la recherche d'analyse de sons extraterrestres.

5 État actuel du projet

On pourrait penser qu'un tel projet est trop ambitieux pour être réalisé par quatre éleves d'info SPE à l'EPITA, mais non :

Patchwork13 a en fait démarré bien avant que nous décidions de le soutenir cette année, sous forme de projet open-source (sous license GPL) hébergé par SourceForge.net.

Officiellement lancé l'anné dernière par billitch, celui-ci mature depuis quelques années déja dans sa tête. Il est donc inutile de préciser qu'intêret et motivation sont présents.

Actuellement écrit en C plus ou moins standard, le projet actuel se découpe en trois parties :

- pw13 (le noyau) : une librairie permettant d'instancier et connecter des patches, et de faire avancer les données dans ceux-ci.
- pw13_std : une librairie de patches standard qui manipulent quelques types de données simples (eg entiers, flottants, flux).
- pw13_gtk : une interface graphique GTK+ permettant d'utiliser les deux librairies précédentes (et les futures autres librairies de patches) "à la souris".

En plus de ces trois parties, un programme de démonstration, appellé pw13_demo qui permet de tester les fonctionnalités présentes.

Commençons par expliquer quelques concepts que le projet a développés

5.1 Concepts

5.1.1 Patchwork

Structure contenant un graphe de patches. Il enregistre également les types de données que manipulent les patches du graphe.

5.1.2 Classe de patch

Objet permettant de créer un patch.

Il contient les fonctions membres du patch ainsi qu'un constructeur qui enregistre les types de données auprès du patchwork, initialise le patch et le lie à ses fonctions membres grâce à des pointeurs vers fonctions. On l'utilise principalement sous forme de librairie dynamique.

5.1.3 Patch

Objet comportant :

- un tableau d'entrées
- un tableau de sorties
- une ou plusieur fonctions membres, notament celle qui calcule la valeur des sorties en fonction de celles des entrées (pump).

5.1.4 Entrée

Structure dotée d'un pointeur sur une sortie, d'un type de données (pour verifier l'homogénéité), et d'une valeur par défaut qui est utilisée si l'entrée n'est connectée à aucune sortie.

5.1.5 Sortie

Structure contenant une liste des entrées qui y sont connectées, un type de données (pour l'homogénéité) et une donnée (à laquelle accèdent les patches connectés pour calculer les valeurs de leurs sorties, ou opérer un effet de bord).

5.1.6 Donnée

Union de tous les types simples que le C standard fournit.

On peut notemment y mettre un pointeur vers un type structuré, une fonction, un patch...

Le type n'est pas enregistré dans la donnée puisqu'on ne peut connecter que des entrées et sorties dont les types sont compatibles. Un type n'est donc qu'un nom donné à une convention respectée par les patches.

5.2 pw13

pw13, la librairie noyau, fonctionne.

Elle peut charger une classe de patch depuis un fichier (une librairie dynamique), créer un patchwork, instancier des patches dedans, les connecter puis faire fonctionner le tout (pomper les données à travers les patches).

5.3 pw13_std

La librairie de patches standard est en fait une collection de classes de patches.

Elle doit fournir les manipulations de bases sur quelques types de données simples. Loin d'être complète, elle contient déjà quelques classes, qui sont ordonnées selon le type auquel elles se ratachent le plus :

- entier : addition, soustraction, division euclidienne, factorielle, maximum, écriture dans un flux.
- nombre flottant : addition, soustraction, oscillateur, sinus (désolé splin), écriture dans un flux, affichage (en chiffres) dans un flux.
- flux : stdin, stdout, stderr, tubes (non testé).
- texte : concaténation, écriture dans un flux.
- temps : incrementation, conversion depuis un flottant.

Seules quelques une de ces classes ont été testées, il est donc probable que certaines ne marchent pas.

5.4 pw13_gtk

L'interface graphique pour GTK+.

Encore au stade d'embryon, tout est à (re)faire puisque le peu ayant été réalisé utilise directement les fonctions de GTK+.

Pure folie pour ceux qui connaissent glade (une interface graphique de création d'interfaces graphiques, générant les sources qui utilisent GTK+), et *suicide* pour celui qui connait libglade, une librairie qui permet de charger dynamiquement une interface concue avec glade.

L'intêret est de ne pas avoir a recompiler quoi que ce soit pour modifier l'interface puisque celle-ci est chargée depuis un fichier XML lors de l'execution du programme.

Toute l'interaction entre l'interface, la librairie noyau et les diverses librairies de patches reste à faire.

5.5 pw13_demo

Un petit programme de test, qui fonctionne!

Il appelle les fonctions de pw13 pour créer un patchwork, charger quelques classes, créer quelques patches, les connecter et pomper les données a travers les patches.

Actuellement, il connecte une constante flottante à la frequence d'un oscillateur lui même relié à un patch qui affiche un fottant, à son tour connecté à un patch d'incrémentation du temps.

6 Découpage du projet

Voici comment notre travail sur le projet sera découpé.

6.1 Noyau

Par billitch.

Si le noyau fonctionne actuellement, il reste toutefois des choses à revoir :
- le parcours du graphe est fait récursivement, ce qui impose une limite sur sa profondeur. Un algorithme itératif sera évidemment plus adapté.
- les listes d'entrées et de sorties d'un patch sont en fait des tableaux alloués lors de la création du patch. Une implémentation par des listes chainées rendrait abordable l'ajout et la suppression d'entrées et sorties en cours de calcul.
- les types de données ne sont compatibles que s'ils sont identiques. On pourrait permettre de faire un alias d'un type, d'indiquer la compatibilité entre deux types, et même gérer un polymorphisme !

6.2 Librairie standard

Par tout le monde.

La librairie standard contiendra les fonctions de base mais très importantes (addition, multiplication, sinus, ...). Pour l'utilisateur, elle permettra de se familiariser avec le logiciel et restera omniprésente pour ajouter des calculs simples au traitement.

Nous comptons tous travailler dessus pour nous familiariser avec le code deja present (sauf pour billitch). La création des opérations sur les types simples (flottant, entier, texte...) sera donc de la partie lorsque nous nous concentrerons sur cette librairie.

6.3 Interface graphique

Par Billich & loucha_m

l'interface graphique, essentielle a une bonne comprehension et une simple utilisation du patchwork13, sera composée de plusieurs parties.

6.3.1 La fenêtre principale

Elle sera composée
- D'une barre des taches présentant les differentes fonctionnalités utilisateur. On y verra aparaître des outils de base tels que charger un patchwork, sauvegarder.. et les differents outils graphiques nécessaire a l'élaboration d'un patchwork.
- D'une boite à patch, contenant les différents patch utilisables a cet instant pour l'élaboration d'un patchwork. On pourra voir les details de chaque patch et y choisir le plus adequat.

– D'une fenêtre de log, permetant d'afficher pas à pas le suivi du patchwork et en cas d'erreurs les afficher afin de les corriger aisément.

6.3.2 La fenêtre d'un patchwork

La fenêtre du patchwork est l'espace de travail où l'on peut lier, grâce a la souris, les patchs entre eux. On y verra le graphe des patchs, qui sera representé comme l'utilisateur le souhaite. Il lui suffira de deplacer les patchs pour qu'il adapte la forme de son patchwork à sa representation mentale.

6.3.3 La fenêtre d'un patch

Chaque patch sera représenté dans la fenêtre de son patchwork par un widget GTK+ où toutes les propriétés du patches seront accessibles.
Cela inclue les entrées et sorties du patch, que l'on pourra connecter à celles d'un autre patch en tirant à la souris une ligne entre les deux.

Le problême se corse lorsque l'on veut gérer une interface propre à une classe de patches. Par exemple on aura un patch oú l'utilisateur peut taper un nombre au clavier dans la fenêtre du patch, qui renverra alors cette constante en sortie.

Pour cela, il faut que les patches qui veulent une interface graphique fournissent une fonction pour créer cette interface.
Cette fonction prendra en paramètre le widget du patch et y créera les widgets propres au patch.

6.4 Librairie d'entrée/sortie audio

Par loucha_m

Dans notre objectif final du patchwork13, les entrées et sorties sonores sont primordiales pour sentir l'interactivité avec la machine. Entrées et sorties sonores, cela implique une communication avec la machine et plus précisement la carte son. Pour ce faire, la librairie utiliée sera la SDL. Un moment de doute nous est venu lors de ce choix. En effet, Fmod, déja utilisée par plusieurs membres du groupe à ete notre premier choix. Mais la SDL est, au niveau du code assez similaire. Ce n'est pourtant pas cela qui nous a decidés. L'important pour nous est que la SDL est open source est ne demande en aucun cas de payer pour l'utiliser, a l'inverse de Fmod.

Cette partie, de communication sonore avec la machine, m'intéresse par son côté multimedia et interactif. L'utilisation l'an dernier de la librairie Fmod etait tres intéressante et pouvoir changer de temps en temps, he ben ! ça fait pas de mal !

6.5 Librairie de traitement et synthèse audio

Par billitch et Splin.

Faisant tous les deux de la musique sur ordinateur, nous sommes particulièrement intéressés par la synthése du son. Créer son propre instrument est

toujours fortement jouissif, et la synthèse audio n'est pas l'origine du projet sans raison.

6.5.1 Synthèse

Nous décomposerons la génération d'un son de la maniére suivante :
- Les patches "synthétiseurs" prennent en paramètre une fréquence (donc un nombre flottant) et éventuellement d'autres paramétres.
- Un patch pour dessiner des courbes de fréquences.
- On fournira un patch pour convertir une note en fréquence, et réciproquement.
- Divers patches de génération de notes.
- On pourra eventuellement faire des patches de transformation des notes (eg transposition, changement d'octave, transformation majeur-mineur)
- Il faut ajouter à ça des patches d'enveloppe (le volume des notes jouées)

6.5.2 Traitement

La librairie permettra de traiter le signal audio, nous proposons d'implémenter nous même les effets suivants :
- delai (echo)
- filtres passe haut, passe bas, passe bande
- dynamiques : limiteur, compresseur

6.6 Librairie OpenGL

Par Broen

Nous comptons créer quelques patchs permettant d'utiliser les fonctionnalites de la librairie OpenGl. Nous ne pourrons pas tout apporter a l'utilisateur mais nous essayerons d'ammener un minimum de fonction pour l'aider, a travers des modules.

Par exemple par le biais de plusieurs patchs prenant en parametre diverses données, le processus affichera un cercle, triangle, onde... bien sur il faudra ecrire des types pour gérer la 3D (point, ligne, volume).

La librairie OpenGL s'est naturellement imposée pour gérer cette partie par sa portabilite, sa simplite et sa puissance face a ses concurrents...

J'ai choisi de m'occuper de cette partie pour explorer cette librairie qui me paraît très interessante a connaître. Dans mes deux précédents projets je suis passé à cote et enfin je vais pouvoir la découvrir. Elle me reservira surement dans le futur donc autant commencer dès maintenant.

6.7 Cluster

Par Broen et Splin

Le cluster le permettra d'augmenter les possibilités de ce projet. En effet cela conferera une puissance de calcul et une rapidite d'exécution de l'enchaînement

des patchs si le processus est trop fastidieux a obtenir pour une seule machine.

Pour ce faire, nous utiliserons la librairie standard du C pour gérer le clustering. La raison de ce choix est que nous voulions coder directement avec les possibilités de base du langage C. Une autre raison est que les librairies qui nous interessaient possèdent des contraintes, telles que la portabilité, ou encore le protocole reseau utilisé, qui les ont exclues.

La méthode utilisée sera client-serveurs : les serveurs en ecoute, attendent une requete du client et renveront le resultat du travail qui leur aura ete demande. Tant au niveau client que server, il y aura 3 threads : un de réception, un d'envoi et un de calcul.

Interets de Broen et Splin : ayant en commun un interêt pour le reseau nous avons decidé de nous occuper de cette partie. Tout les deux nous avons realisé la partie reseau de nos projets de Sup respectif et Broen celui de Spe également. De plus, travailler directement avec les sockets nous paraît bien plus enrichissant qu'utiliser une librairie. La raison certainement la plus pertinente est que nous pensons fortement â travailler dans ce domaine dans l'avenir.

6.8 Maintenance et programmes d'installation

Tout le monde.

Le projet est actuellement géré avec autoconf et automake, ce qui le rend facile à porter sur la plupart des UNIX et même windows grace à MinGW (Minimalist GNU for Windows) et MSYS qui permettent de compiler avec gcc les applications UNIX en programme Windows natif.

Nous fournirons tout de même un programme d'installation pour Mac OS X et Windows, afin de rester accessible à tout public.

6.9 Documentation

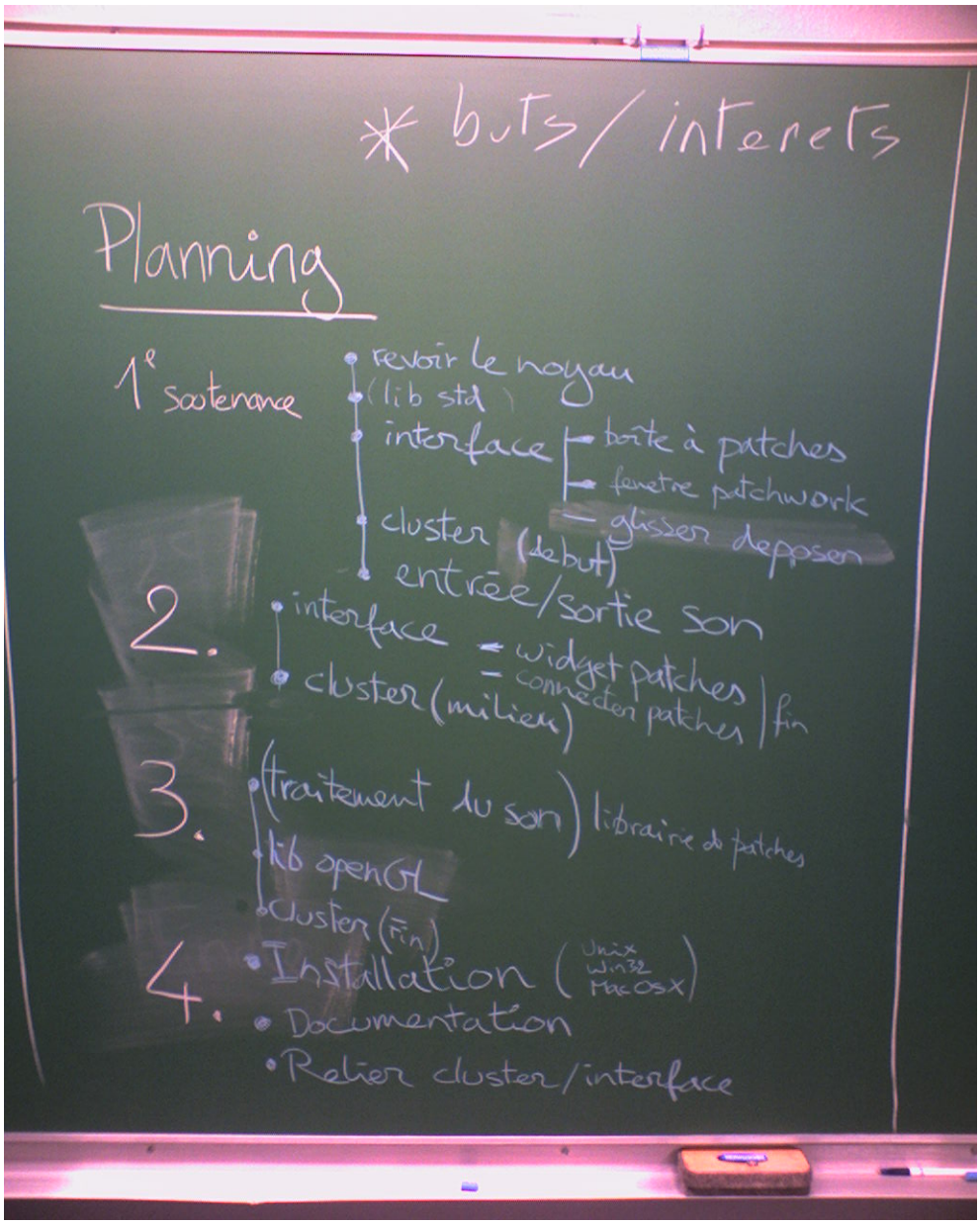
Tout le monde.

Chaque partie du projet devra être documentée en même temps que le développement par la personne s'occupant de la partie, puisque c'est elle qui connait le mieux le sujet.

Doxygen, un programme générant une documentation à partir du code source, s'averera surement utile pour documenter les librairies.

7 Logistique

7.1 Planning ?



7.2 Planning !

Soutenance 1	Soutenance 2	Soutenance 3	Soutenance Finale

– Revoir le noyau
– Interface
  – Boite a patch
  – Fenetre patch-work
  – Glisser-deposer
– Cluster (debut)
– Entree|Sortie son
	– Interface
	  – Widget patches
	  – Connector patches
	– Cluster (milieu)
		– Lib de patches (traitement du son)
		– Lib OpenGL
		– Cluster (Fin)
			– Instalation (multi-plateformes)
			– Documentation
			– Relier Cluster et Interface

7.3 Couts matériels

- un powerbook 13", trois PC.
- 523 machines en réseau pour tester le cluster
- Plateformes de développent
- Environ 600 litres de biere, 160 pizzas, 160kg de frites ...
  En resumé, ca va nous couter cher.

8 Références

- EPITA – Ecole Pour l'Informatique et les Techniques Avancées
  http ://www.epita.fr/
- Patchwork13 ! – Synthèse modulaire universelle
  http ://patchwork13.sourceforge.net/
- OpenGL – Open Graphic Library
  http ://www.opengl.org/
- SDL – Simple DirectMedia Library
  http ://www.libsdl.org/

Synthétiseurs modulaires (audio) :

- Buzz – Populaire et gratuit
  http ://www.buzzmachines.com/
- MAX/MSP – Sûrement le plus puissant synthétiseur modulaire à ce jour
  http ://www.cycling74.com/products/maxmsp.html
- jMax – Le clone java open-source de Max/MSP, développée par l'IRCAM
  http ://freesoftware.ircam.fr/rubrique.php3 ?id_rubrique=2

9 Conclusion

Outre l'aspect travail, nous sommes un groupe qui nous connaissons depuis la Sup mais nous n'avions pas été dans les mêmes groupes jusque là. Ceci va nous permettre de coder dans une ambiance conviviale et détendue (Dis moi une question : Qui est détentu ?). A notre ce sens ceci jouera forcément sur le travail et sur l'avancement du projet.

Passons maintenant a l'autre aspect. Collectivement nous avons tous touché de plus moins loin au milieu de la musique (composition, mix, instrument ...) ou de la video. Mais le fait que le projet ait une dimention presque infinie et que de nombreuses personnes extérieures soient intéréssées par l'idée nous motive grandement. De plus, le projet est sous GPL ce qui permettra aux personnes extériereures de développer des patchs et former une communauté par la suite ce qui serait a notre sens une reussite sur tous les plans.

MERCI.