Normalització del volum d'arxius de so en format MP3

Autor: David Pujadas Costafreda
Consultor: Antonio Bonafonte Cávez

Universitat Oberta de Catalunya

Data: 27/06/2002

ÍNDEX

PRELIMINARS

Es pretén construir una aplicació que permeti normalitzar el volum d'una col·lecció de fitxers de so en format MP3. Aquesta aplicació s'anomenarà *normalitzar*.

Degut a la dificultat que implica la realització de la totalitat del projecte, s'ha decidit dividir-lo en diverses parts que seran desenvolupades per diferents estudiants

En aquest treball de fi de carrera es dissenyarà i implementarà la part de l'aplicació destinada a extreure la informació dels fitxers MP3 (components freqüencials, estructura dels *frames*, etc) i la part destinada creada a crear el nou fitxer MP3 amb el nou guany normalitzat.

OBJECTIUS

Aquest projecte s'emmarca dins l'assignatura Treball de fi de carrera (TFC) de l'enginyeria tècnica en informàtica de sistemes de la Universitat Oberta de Catalunya. Seguint les directrius de l'assignatura, aquest TFC té els següents objectius:

- Analitzar l'estàndard *MPEG Layer III*
- Veure com es codifica el volum del so en aquest format
- Analitzar com cal modificar-lo perquè tots els fitxers sonin igual
- Realitzar una part de la implementació
- Elaboració de la memòria
- Presentació virtual amb el programa *Power Point* de Microsoft

1. L'estàndard MPEG Layer III

1.1 Introducció

L'esquema per a la codificació d'àudio *MPEG Layer III* ja ha celebrat el seu 10è aniversari, després de ser estandarditzat el 1991. En els primers anys, l'esquema era bàsicament utilitzat per *codecs* d'aplicacions d'estudi. El 1995, l'*MPEG Layer III* va ser escollit com el format d'àudio pel sistema de difusió digital via satèl·lit desenvolupat per WorldSpace. Aquest va ser el primer pas en el gran mercat. El segon pas va arribar ben aviat, degut a la utilització d'Internet com a sistema de distribució electrònic de música. Amb això, la proliferació de material d'àudio codificat amb *MPEG Layer III* (també conegut com a MP3) ha experimentat un creixement exponencial des de 1995. De fet, a principis de 1999, la paraula ".mp3" era la búsqueda més popular a la Web.

Tot va començar a mitjans de la dècada dels 80, a l'Institut Fraunhofer a Alemanya, on van començar a treballar en un nou sistema de codificació d'àudio d'alta qualitat i baix *bit-rate*. El 1989, l'institut va aconseguir la patent de l'MP3 a Alemanya i pocs anys després el van enviar a l'Organització Internacional d'Estàndards (ISO), que el va incorporar dins l'especificació MPEG-1.

Fraunhofer també va crear el primer reproductor MP3 a principis dels 90. Però no va ser fins el 1997 que va aparèixer el Motor de Reproducció MP3 d'AMP, que amb la interfície creada per uns estudiants universitaris (el famós *WinAmp*) va començar la bogeria de l'MP3: 'amics' de la música de tot el món van començar a oferir música protegida per drets d'autor gratuïtament.

1.2. Codificació MP3

1.2.1 Característiques de la codificació

A continuació presentem les característiques bàsiques i alguns detalls necessaris per entendre les implicacions de les diferents opcions de codificació en la qualitat del so.

1.2.1.1. Flexibilitat

Amb l'objectiu que es pogués utilitzar en una gran varietat d'escenaris, MPEG va definir una representació de dades que inclou un gran nombre d'opcions.

1.2.1.2. Mode de funcionament

MPEG-1 àudio funciona tan en senyals mono com estèreo. Una tècnica anomenada codificació *joint stereo* es pot usar per a aconseguir una codificació més eficient dels canals esquerre i dret d'un senyal estereofònic. Els diferents modes de funcionament són:

- Canal únic (*single channel*)
- Canal doble (*dual channel*, dos canals independents per contenir, per exemple, versions en diferents idiomes)
- *Stereo*
- *Joint stereo*

1.2.1.3. Freqüència de mostreig

La compressió d'àudio MPEG funciona amb diferents freqüències de mostreig. MPEG-1 defineix la compressió d'àudio a 32 kHz, 44.1 kHz i 48 kHz.

MPEG-2 ho estén a les seves respectives meitats (16 kHz, 22.05 kHz i 24 kHz). MPEG-2.5 és el nom d'una extensió propietària de la capa III (*layer III*) desenvolupada per l'IIS Fraunhofer, que introdueix les freqüències de 8 kHz, 11.05 kHz i 12 kHz.

1.2.1.4. Bit-rate

MPEG àudio no només treballa a un ratio de compressió fix. La selecció del *bit-rate* de l'àudio comprimit es deixa, dintre d'uns límits, a l'elecció del creador o de l'usuari del codificador MPEG. Per la capa III, l'estàndard defineix un rang de *bit-rates* que va des de 8 kbit/s fins a 320 kbit/s. A més, els descodificadors han de suportar el canvi de *bit-rate* d'un quadre (*frame*) a l'altre. Això combinat amb la tecnologia de reserva de bits permet tant la codificació amb *bit-rate* variable com la codificació amb *bit-rate* constant a un valor fixat dins dels límits imposats per l'estàndard.

1.2.1.5. Normatiu vs Informatiu

Una característica molt important dels estàndards MPEG és el principi de minimitzar la quantitat d'elements normatius dins de l'estàndard. En el cas de l'MPEG àudio, això significa que només la representació de les dades (per exemple, el format de l'àudio comprimit) i el descodificador són normatius.

1.2.2. L'algoritme

Els següents paràgrafs descriuen l'algoritme de codificació de la capa III, així com els blocs bàsics del codificador.

La figura 1 mostra el diagrama de blocs d'un codificador *MPEG Layer III* típic.

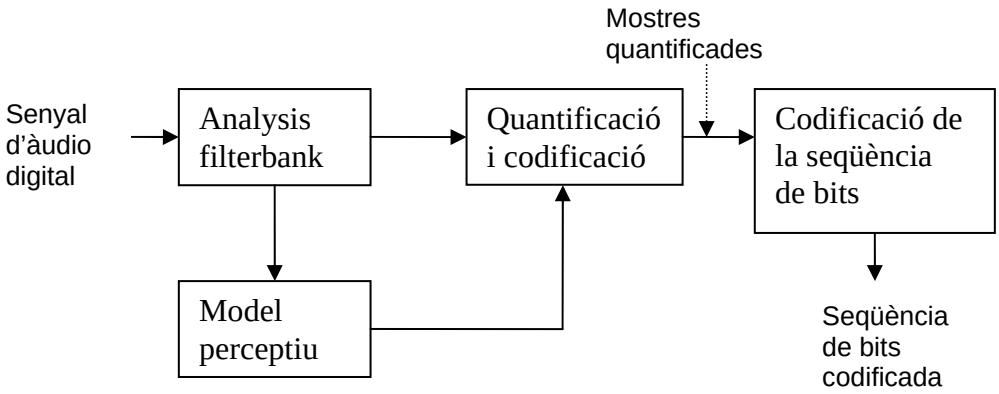

Figura 1

1.2.2.1. Analysis filterbank

El filtre utilitzat en MPEG Layer III pertany a la classe dels filtres híbrids. Està construït connectant en cascada dos tipus diferents de filtre: primer, un filtre polifàsic (com els utilitzats a *Layer I* i *Layer II*) i segon, un filtre de Transformació Discreta del Cosinus Modificat (MDTC). El filtre polifàsic aconsegueix l'objectiu de fer la capa 3 més semblant a les dues primeres. La subdivisió de cada banda de freqüències polifàsiques en 18 sub-bandes més petites incrementa el potencial per eliminar la redundància, assolint la millor eficiència de codificació en senyals tonals. Com a resultat de la resolució a altes freqüències, la senyal d'error es pot controlar millor, permetent que el llindar es pugui afinar amb més precisió.

1.2.2.2. Model perceptiu

El model perceptiu principalment determina la qualitat de la implementació d'un codificador donat. Des que es va publicar la part informativa de l'estàndard, s'ha realitzat una bona quantitat de treball addicional en aquesta part del codificador. El model perceptiu tant pot utilitzar un filtre separat com combinar el càlcul de valors d'energia amb el filtre principal. La sortida del model perceptiu consisteix en una sèrie de valors pel llindar d'emmascarament o de soroll permès per cada part de la codificació. En la capa III, aquestes parts de la codificació són aproximadament equivalents a les bandes crítiques de l'oïda humana. Si el soroll de quantificació es pot mantenir per sota del llindar d'emmascarament per cada part de la codificació, aleshores el resultat de la compressió no es podrà distingir de la senyal original.

1.2.2.3. Quantificació i codificació

La solució més comuna en el procés de quantificació i codificació en un codificador de capa 3 consisteix en un sistema de dos bucles anidats. El sistema de quantificació fa que els valors més alts es codifiquin amb menys precisió, però s'afegeix una mica de soroll en el procés. Els valors quantificats es codifiquen mitjançant la codificació de Huffman. Per tal d'adaptar el procés de codificació a les diferents senyals musicals, s'escull la taula Huffman òptima d'entre un conjunt d'opcions. La codificació Huffman funciona amb parelles, i en el cas de números molt petits, en grups de quatre. Per aconseguir una millor adaptació a les característiques de la senyal, es poden escollir diferents taules de Huffman per cada part de l'espectre. Com que la codificació Huffman és, bàsicament, un mètode de codi de longitud variable i com que s'ha de fer un tractament del soroll per mantenir-lo per sota d'un llindar, s'aplica un guany global abans de la quantificació. El procés per trobar el guany òptim es porta a terme per part de 2 bucles anidats:

- Bucle intern (*rate loop*)

  La codificació Huffman assigna paraules més curtes als valors més freqüents. Si el nombre de bits resultant és més gran que el nombre de bits disponibles per codificar un determinat bloc de dades, es pot corregir aplicant el guany global al resultat, comportant valors més petits. Aquesta operació es repeteix fins que els valors obtinguts són prou petits.

- Bucle extern (*noise control loop*)

  Aquest bucle permet mantenir el soroll afegit per sota del llindar permès. Com que aconseguir un nivell de soroll més petit requereix més passos de quantificació (i, per tant, un major *bit-rate*), el bucle d'ajustament del *bit-rate* s'ha de repetir cada vegada que s'utilitza un nou factor.

1.2.3. Diferències entre codificadors

Els estàndards MPEG no imposen la implementació del codificador d'àudio. En un cas extrem, algú podria no implementar el model perceptiu, decidir no utilitzar el bucle extern i fer un bucle intern molt simple. Un codificador com aquest seria molt ràpid (segurament més ràpid que qualsevol dels productes actuals), compliria amb l'estàndard, fins i tot podria produir so d'alta qualitat per algunes senyals, però sonaria molt malament en una selecció de música prou gran. Mentre que un projecte com aquest seria molt senzill d'implementar, és molt més difícil de construir un codificador que ofereixi una alta qualitat d'àudio en tot tipus de música. En MPEG, les proves sempre han intentat verificar el comportament dels codificadors en els pitjors escenaris. Tot i així, els codificadors actuals d'MP3 ofereixen una alta qualitat de so a *bit-rates* prou baixos.

1.2.3.1. Com mesurar la qualitat del so

Mesurar la qualitat del so de *codecs* d'àudio perceptius ha esdevingut un autèntic art en els darrers anys. Bàsicament, hi ha 3 mètodes: test d'escolta, mètodes de mesura objectius i tècniques de mesura perceptives.

A dia d'avui, els tests d'escolta a gran escala són l'únic mètode per comparar el comportament de diferents algoritmes de codificació. L'ITU-R ha desenvolupat un elaborat conjunt de normes per dur a terme aquest tipus de test, que intenten portar els codificadors a les pitjors condicions possibles. Així, els codificadors s'han d'anar afinant poc a poc per tal de satisfer els requeriments dels oïdors més experts.

Per altra banda, s'ha intentat mesurar la qualitat d'un codificador mirant paràmetres com la relació senyal-soroll o l'amplada de banda de la senyal descodificada. Una altra opció és mirar la senyal resultant buscant certs aspectes de la senyal d'entrada, com ara transitoris o senyals multi-to. Els

resultats d'aquests tipus de test poden dir molt als experts sobre el *codec* que s'està provant, però és molt perillós basar-se només en aquests resultats.

Per últim, l'aplicació de models psico-acústics a la predicció de la qualitat del so ha avançat molt en els darrers anys i, en alguns casos, ja està substituint els tests d'escolta.

1.2.3.2. El mite de l'amplada de banda

Els informes sobre les proves efectuades a codificadors sovint mencionen l'amplada de banda de la senyal d'àudio comprimida. En molts casos això és degut a malentesos entre l'oïda humana per una banda i estratègies de codificació per l'altra.

És cert que els individus poden sentir sons aïllats per sobre dels 20 kHz, però diversos estudis conclouen que és molt difícil diferenciar contingut musical amb continguts per sobre dels 20 kHz de música amb el límit situat als 16 kHz. Per tant, sembla raonable limitar la resposta freqüencial d'un codificador MP3 als 16 kHz.

Per altra banda, el fet que un codificador produeixi so amb una amplada de banda més gran no implica necessàriament que soni millor. Els bits utilitzats per millorar la resposta freqüencial no es poden utilitzar per produir un so més clar a baixes freqüències.

1.2.3.3. Provant diferents *bit-rates*

Per tal d'aconseguir l'equilibri entre els requeriments del model perceptiu i el *bit-rate* disponible, els paràmetres de codificació s'han de reajustar si el codificador és executat a diferents *bit-rates*. Aquests processos de prova formen bona part de l'esforç que cal dedicar al desenvolupament d'un codificador MP3.

1.3. Descodificació MP3

El descodificador accepta una seqüència de bits d'àudio comprimit, descodifica les dades i utilitza aquesta informació per produir una sortida d'àudio digital.

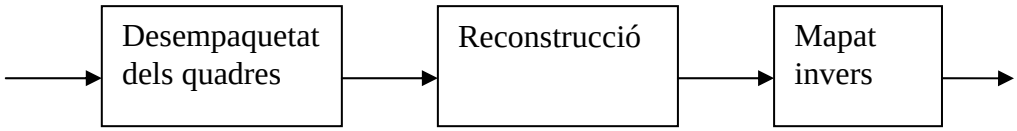

Figura 2

Les dades de la seqüència de bits alimenten el descodificador. El bloc de desempaquetat i descodificació de la seqüència de bits fa la detecció d'errors si s'ha aplicat aquesta funcionalitat en el codificador. El bloc de reconstrucció reconstrueix la versió quantificada de les mostres mapades. El mapat invers transforma aquestes mostres altra vegada en uns senyal PCM uniforme.

1.4. Altres estàndards MPEG

MPEG, un grup de treball formalment conegut com a ISO/IEC JTC1/SC29/WG11, però més conegut pel seu sobrenom, va ser creat per ISO/IEC el 1988 per tal de desenvolupar estàndards genèrics per la representació de gràfics en moviment, l'àudio associat i la seva combinació. Des d'aleshores, MPEG s'ha encarregat de l'estandardització de tècniques de compressió de vídeo i àudio. En un principi, el seu objectiu era la codificació de vídeo juntament amb el seu àudio per emmagatzemar-ho en format digital. Mentrestant, la codificació d'àudio ha trobat el seu propi camí en diferents aplicacions:

- Difusió d'àudio digital
- So per la televisió digital
- *Streaming* a través d'Internet
- Reproductors portàtils
- Emmagatzemament i intercanvi de fitxers de música entre ordinadors

1.4.1. MPEG-1

MPEG-1 és el nom de la primera fase del treball d'MPEG, que va començar el 1988. Aquest treball va finalitzar amb l'adopció de l'estàndard ISO/IEC IS 11172 a finals de 1992. La part de codificació d'àudio d'aquest estàndard (IS 11172-3) descriu un sistema de codificació genèric, dissenyat per assolir les demandes de diverses aplicacions. MPEG Audio consisteix en tres mètodes de funcionament anomenats "capes", amb complexitat i prestacions creixents, anomenades *Layer-1*, *Layer-2* i *Layer-3*. Aquesta última, amb la major complexitat, va ser dissenyada per proporcionar la qualitat més alta a baixos *bit-rates* (al voltant de 128 kbit/s per una senyal estèreo).

1.4.2. MPEG-2

MPEG-2 denomina la segona fase d'MPEG. Va introduir molt conceptes nous a la codificació de vídeo, incloent suport per senyals de vídeo entrellaçat. La principal àrea d'aplicació de l'MPEG-2 és la televisió digital. L'estàndard d'àudio MPEG-2 (IS 13818-3) va finalitzar el 1994.

1.4.3. MPEG-2 AAC

A principis de 1994, algunes proves de verificació van demostrar que nous algoritmes de codificació, sense compatibilitat amb MPEG-1, prometien una millora considerable en l'eficiència. Com a resultat, es va definir un nou element de treball que finalment va portar a la definició d'un nou estàndard, la Codificació Avançada d'Àudio MPEG-2 (AAC, *Advanced Audio Coding*). L'estàndard es va finalitzar el 1997 (IS 13818-7). AAC és un esquema de codificació d'àudio de segona generació per la codificació genèrica de senyals stereo i multi-canal, suportant freqüències de mostreig d'entre 8 kHz i 96 kHz i entre 1 i 48 canals d'àudio.

1.4.4. MPEG-3

Originalment, MPEG havia planificat definir la codificació de vídeo per aplicacions de televisió d'alta definició (HDTV) en una fase posterior, que s'havia d'anomenar MPEG-3. Més tard es van adonar que les eines desenvolupades per l'MPEG-2 podien assolir els requeriments de la televisió d'alta definició, i l'MPEG va desestimar el pla de desenvolupar un estàndard especial MPEG-3. Sovint *MPEG-1 Layer 3* (MP3) es malanomena MPEG-3.

1.4.5. MPEG-4

MPEG-4 intenta esdevenir el següent gran estàndard del món de la multimèdia. La primera versió va finalitzar a finals de 1998 (IS 14496-3), i la segona a finals de 1999. A diferència de MPEG-1 i MPEG-2, l'èmfasi de

l'MPEG-4 és en les noves funcionalitats més que no pas una millor eficiència en la compressió. Terminals d'usuari mòbils, accés a bases de dades, comunicacions i nous tipus de serveis interactius seran les principals aplicacions de l'MPEG-4. El nou estàndard facilita la creixent interacció entre els móns de la informàtica, les telecomunicacions i els mitjans de masses (televisió i ràdio). Els algoritmes del MPEG-4 permeten codificar àudio des de 2 kbit/s fins a 64 kbit/s per canal. La codificació a majors *bit-rates* es porta a terme mitjançant AAC.

1.4.6. MPEG-7

A diferència de MPEG-1, MPEG-2 i MPEG-4, MPEG-7 no defineix algoritmes de compressió. MPEG-7 és una representació estàndard de continguts per a la búsqueda, filtrat, manejament i processament d'informació multimèdia.

1.5. L'èxit d'MP3

MPEG Layer-3 ha esdevingut la principal eina per l'intercanvi de música a través d'Internet. Considerant les raons, els següents factors han estat de gran ajuda.

1.5.1. Estàndard obert

MPEG està definit com un estàndard obert. L'especificació està disponible per tothom. Tot i que hi ha algunes patents cobrint alguns aspectes de la codificació i descodificació d'àudio MPEG, els propietaris de les patents han declarat que llicenciaran les patents sota termes raonables per tothom. Codi font d'exemple públic està disponible per ajudar els implementadors a entendre el text de l'especificació. Com que el format està ben definit, no hi ha problemes d'interoperabilitat entre els equipaments de diferents fabricants.

1.5.2. Disponibilitat de codificadors i descodificadors

Codificadors i descodificadors hardware i software basats en processadors digitals de senyals (DSP) han estat disponibles des de fa una colla d'anys (primer gràcies a la demanda per l'ús professional en la difusió).

1.5.3. Tecnologies de suport

Mentre que la compressió d'àudio és vista com la principal tecnologia, altres tecnologies associades han contribuït al 'boom' de l'MP3, com per exemple:

- L'ampli ús de targetes de so en els ordinadors
- Els ordinadors són prou ràpids com per executar descodificadors (i fins i tot codificadors) en temps real

- Ràpid accés a Internet (ADSL, cable, ...)

- La disponibilitat de CD-ROM i gravadores de CD

En poques paraules, MP3 va tenir la sort de ser la tecnologia adequada en el moment oportú. Mentrestant, la recerca en la codificació d'àudio perceptiva no s'ha aturat, i *codecs* amb un millor ràtio de compressió han sortit a la llum. D'aquests, l'AAC ha estat desenvolupat com a successor de l'MPEG-1.

1.6. El futur d'MP3

Sempre és difícil de dir quant de temps una determinada tecnologia serà vigent, però tenint en compte tota la infrastructura que s'ha muntat al voltant de l'MP3, és possible que aquest estàndard duri força temps entre nosaltres. Hi ha altres tecnologies de compressió millors, però la naturalesa relativament oberta de l'MP3 és la clau del seu èxit.

Aquesta situació, però, pot canviar. Des del moment que l'Institut Fraunhofer va anunciar que demanaria drets de patent per cada transmissió a Internet que inclogués un fitxer MP3, es va posar en marxa un projecte de codi obert anomenat Ogg Vorbis. Aquest grup està intentant desenvolupar una tecnologia de codificació d'àudio oberta i lliure de patents amb tots els beneficis de l'*Open Source*.

Com l'MP3, el format Ogg Vorbis determinarà quines parts d'un fitxer de so no són audibles i les descartarà durant el procés de codificació. A diferència de l'MP3, serà un format totalment obert, al qual podrà contribuir tothom.

Mentrestant, el coneixement i la utilització del format MP3 no para de créixer, i ha esdevingut una arrelada indústria a la Web sense un límit a la vista, sobretot gràcies a la seva senzillesa: tot el que fa falta és un ordenador no massa potent per tenir tota la música a la teva disposició.

2. Normalització del volum

2.1. El problema

No tots els CDs de música sonen igual de fort. El volum percebut dels fitxers MP3 és encara més variable.

Mentre que alguns estils musicals requereixen que algunes cançons haurien de sonar més altes que d'altres, el volum d'un CD determinat té més a veure amb l'any en què va ser creat o amb un desig del productor que amb l'efecte emocional que es volgués donar.

Si a tot aquest caos li afegim la qualitat inconsistent de la codificació MP3, la gran varietat de codificadors existent i la inacabable quantitat de fitxers disponibles a Internet, no ens ha d'estranyar que una reproducció aleatòria de la nostra col·lecció de música en aquest format ens mantingui més pendents del control del volum que de qualsevol altra cosa.

Com es pot aconseguir, doncs, una llista de reproducció consistent, una que no ens faci estar constantment pendents del mesurador del volum?

Una manera d'aconseguir això és normalitzant els fitxers MP3.

2.2. La solució

Normalitzar el volum d'un conjunt de fitxers musicals consisteix a analitzar la forma d'ona de la música digital, extreure els pics i les valls de certs fitxers, fer pujar tots els punts dels fitxers amb menys amplitud i finalment crear una experiència auditiva més suau.

Una primera idea per dur a terme això seria fer que tots els fitxers assolissin el mateix volum màxim. Per fer-ho, només cal multiplicar cada mostra per un valor constant (multiplicar totes les mostres pel mateix valor és amplificar –si el valor és positiu- o atenuar –si el valor és negatiu- i és el que fan els amplificadors).

Per tant, la primera cosa que podríem fer seria recórrer el fitxer, buscar el valor màxim i calcular el factor que el porta al màxim (255 si es representa en 8 bits, 65535 si es representa amb 16 bits –com en els CDs-, ...).

Però ens podem trobar que en un fitxer determinat hi ha algun valor molt més gran que els altres (soroll, so imprevist, ...) que farà que la possible amplificació quedi en poca cosa. En aquests casos pot resultar millor prendre les mesures amb més d'una mostra; és a dir, recórrer el fitxer a intervals (per exemple, de 5 en 5 mostres) i calcular la mitjana de les mostres. Ara, la normalització farà que aquests valors mitjans assoleixin el valor que nosaltres determinem.

Una última opció consistirà a calcular la mitjana de tot el fitxer i fer que aquesta assoleixi el valor que determinem.

En aquests casos en què treballem amb mitjanes, cal anar amb compte que algunes mostres assoliran valors més grans que el valor màxim representable. Caldrà, doncs, establir algun mecanisme perquè aquestes mostres no sobrepassin aquest valor.

Tot això sembla molt senzill quan es tracta de fitxers que representen la música en forma dels valors que pren l'ona al llarg del temps, com pot ser el format .WAV.

Però quan parlem de MP3, en què la informació fa referència als components freqüencials de l'ona i, a més, es presenta en forma comprimida, la cosa ja no és tan senzilla.

De totes formes, hi ha diverses maneres de tractar el problema i les passem a veure tot seguit.

2.2.1. Descodificació / recodificació

Si hem comentat que la forma més fàcil de normalitzar el volum d'un fitxer és a partir de la seva representació temporal, sembla lògic pensar que ens podem basar en aquesta representació per normalitzar els nostres fitxers MP3.

El mètode consisteix en descodificar la informació recollida en el fitxer MP3 per tal de transformar-la en un fitxer WAV. Un cop feta la descodificació, ja podem treballar en aquest format (més senzill) per tal de normalitzar-ne el volum. Un cop feta la feina bruta, ja només cal tornar a codificar el fitxer WAV seguint l'estàndard per tal de tornar a obtenir un fitxer MP3 amb la mateixa cançó que abans però amb el nivell de volum desitjat.

Per tal de transformar un fitxer MP3 a WAV, podem fer servir diverses aplicacions disponibles (per exemple, el mateix reproductor WinAmp disposa d'un *plug-in* que permet guardar la sortida a disc en format WAV) o utilitzar alguna de les llibreries creades a tal efecte (per exemple, MAD –MPEG Audio Decoder-, que proporciona una sortida PCM per dades codificades amb MPEG-1 o MPEG-2).

El procés de normalització consistirà en realitzar els càlculs explicats en el capítol anterior (trobar el màxim, calcular les mitjanes, ...).

I pel que fa al procés de codificació, podem fer servir qualsevol dels codificadors disponibles al mercat (alguns de gratuïts –per exemple, LAME- i d'altres de propietaris) o decidir-nos a crear el nostre propi codificador (podem utilitzar tot el codi d'exemple que el propi MPEG posa a disposició dels desenvolupadors).

2.2.1.1. Avantatges

El principal avantatge d'aquest mètode és que és molt més senzill treballar amb fitxers WAV (o PCM) que no pas MP3 o qualsevol altre format que no sigui temporal.

A més, ja hi ha molta feina feta en aquest aspecte (molta d'ella sota llicència GPL) i ens serà possible agafar idees d'altres projectes per tal d'implementar la millor solució possible.

2.2.1.2. Inconvenients

És el procediment que presenta més inconvenients.

El principal és que amb el procés de descodificació i recodificació estem introduint soroll en el resultat final (no hem d'oblidar que la codificació MP3 elimina tots aquells elements que 'creu' que són imperceptibles), amb la pèrdua de qualitat que això suposa.

A més, el temps invertit és molt gran, ja que tant la descodificació com, sobretot, la codificació, requereixen un temps considerable (sobretot en ordinadors no massa potents).

I per acabar, és molt probable que s'hagin d'utilitzar productes de tercers (sobretot pel que fa referència al codificador), i caldrà cenyir-nos als seus requeriments, amb la pèrdua de flexibilitat i autonomia que això suposa.

2.2.1.3. Normalize

Normalize és una eina per ajustar el volum de fitxers d'àudio a un nivell de volum estàndard. És útil per tasques com crear CDs a partir de cançons de diferents àlbums o col·leccions de fitxers MP3, en els quals els diferents volums de gravació dels diversos àlbums pot variar molt de d'una cançó a una altra.

Va ser creada per Chris Vail de la Universitat de Columbia com a projecte de fi de carrera i s'ha convertit en una aplicació alliberada sota llicència GPL.

Normalize utilitza la llibreria MAD per descodificar els fitxers MP3 i convertir-los a format PCM. A més, inclou un *plug-in* pel reproductor XMMS per tal que aquest pugui tractar la informació relativa al volum emmagatzemada en forma de *tags* ID3.

A més, permet la utilització d'altres formats a part del WAV i MP3 gràcies a la utilització de les llibreries SGI.

L'última versió alliberada de l'aplicació és la 0.7.4 (cosa que indica que encara no ha assolit un estat de producció, estat que s'assoleix amb la versió 1.0) que va veure la llum el 23 de maig de 2002.

Hi ha disponible el codi font de l'aplicació preparat per ser compilat a qualsevol màquina que disposi de GNU/Linux, FreeBSD, Solaris o Irix, així com paquets precompilats per les principals distribucions (.rpm i .deb).

El mètode de funcionament del programa és el següent: els volums calculats són amplituds RMS, que es corresponen aproximadament al volum percebut. Agafant l'amplitud RMS d'un fitxer sencer podria no donar la mesura adequada, ja que cançons amb un volum molt baix però amb petits trossos de volum molt alt donarien una mitjana molt baixa i l'ajustament efectuat faria que els trossos de volum més alt sonessin massa forts. És per això que s'agafa el

volum màxim del fitxer i es normalitza d'acord amb ell. Es parteix la senyal en 100 parts cada segon i s'agafa l'energia de la senyal de cada part, amb l'objectiu d'obtenir l'"energia instantània" al llarg del temps. L'"energia instantània" varia massa per obtenir una bona mesura de la màxima "energia sostinguda" de la senyal original. Per tant, executa un algoritme "suavitzador" sobre la senyal original. El punt màxim de la senyal "suavitzada" esdevé una bona mesura de l'"energia sostinguda" màxima del fitxer. Ara ja es pot fer l'arrel quadrada de l'energia per obtenir la màxima amplitud RMS.

2.2.2. Modificació directa del fitxer MP3

Com ja hem dit, el format MP3 no és una representació temporal, sinó freqüencial. És a dir, per cada interval de temps s'especifiquen els components freqüencials del so en lloc de l'amplitud de la senyal.

Però per tal d'amplificar o atenuar el so, el principi hauria de ser al mateix que en una representació temporal: multiplicant cada component freqüencial per un valor constant aconseguirem que la senyal quedi amplificada o atenuada.

Seria com si en un equalitzador (amb el qual podem modificar els tons aguts o els greus) modifiquéssim totes les freqüències amb el mateix factor: el resultat és una amplificació o atenuació de tota la senyal (depenent del signe del factor).

Així doncs, es tracta de mesurar l'energia de la senyal i amplificar tots els components.

2.2.2.1. Avantatges

El principal avantatge d'aquest mètode és que treballem directament sobre el fitxer MP3, no realitzem cap tipus de descodificació i posterior recodificació. Per tant, la qualitat del fitxer resultant serà exactament la mateixa que tingués el fitxer original.

A més, en estalviar processos extra, es produeix un important guany de temps respecte el mètode anterior.

2.2.2.2. Inconvenients

El principal inconvenient és que el format MP3 és molt complex i es fa força difícil treballar-hi directament.

A més, ofereix una gran varietat de variants i es fa molt difícil tenir-les totes en compte.

Per acabar, no és un mètode massa utilitzat per altres aplicacions i, per tant, no en podrem treure massa informació ni idees que ens puguin ajudar en el nostre desenvolupament.

2.2.3. Introduir *tags* ID3

Els mètodes proposats dins ara tenien un tret en comú: la modificació de la informació original (ja sigui amb un procés de descodificació/recodificació o amb una modificació directa de la informació MP3). Això fa que qualsevol petit error en la manipulació de les dades, en la codificació o on sigui produeixi uns resultats desastrosos, irreproduibles.

El mètode que proposem a continuació soluciona el problema, ja que no modifica la informació original, sinó que afegeix certa informació als fitxers referents a la modificació del volum. Aquest concepte s'anomena "MetaData" (o sigui, "dades sobre les dades").

Ja és possible emmagatzemar el títol de la cançó, el nom del cantant, etc. en un fitxer MP3 seguint el format estàndard ID3. El nou estàndard ID3v2 incorpora la capacitat d'emmagatzemar informació relativa a l'ajustament del volum, que hauria de permetre solucionar el problema de la diferència de volum de les col·leccions d'MP3.

Amb tot, no hi ha un estàndard consistent per definir el guany a aplicar amb el qual els codificadors i els reproductors hi estiguin d'acord. A més, no hi ha un mètode automàtic per fixar l'ajustament del volum per cada una de les pistes.

2.2.3.1. Avantatges

El principal avantatge d'aquest mètode és que la informació original no es veu alterada en cap moment, ja que l'únic que es fa és afegir certa informació sobre com tractar la informació continguda en el fitxer. Amb això s'aconsegueix que posteriors modificacions del guany a aplicar al volum no afectin a la qualitat del so.

A més, el mètode es basa en un estàndard obert i amb cada cop més acceptació com és l'ID3v2.

I per acabar, el procés d'extracció de la informació i creació dels *tags* és molt ràpid i permet normalitzar una gran quantitat de fitxers de música amb poc temps.

2.2.3.2. Inconvenients

El principal inconvenient està en la inexistència d'un reproductor que suporti aquest mètode de normalització.

Segur que en el moment que surti algun reproductor (o algun *plug-in* per algun reproductor existent) capaç d'interpretat aquesta informació, el mètode s'estendrà i farà ombra als utilitzats fins al dia d'avui.

2.2.3.3. *Replay Gain*

Replay gain és una proposta d'estàndard publicada el 10 de juliol de 2001, a la qual li queda encara molta feina per fer.

El mètode proposat es basa en calcular dos guanys: un cançó per cançó (anomenat "*radio gain*") que farà que totes les cançons sonin igual de fortes (com si s'estiguessin emetent per la ràdio) i una altre que es basarà en la informació extreta de tot un CD (anomenat "*audiophile gain*") que aconseguirà que un solo de flauta no soni igual de fort que una cançó d'Iron Maiden (cosa ben desitjable, per altra banda). Aquest darrer guany serà ajustable per l'usuari per tal de fer l'experiència auditiva més personalitzada.

Un cop calculats els 2 valors, seran emmagatzemats en un format concret que reserva 2 bytes (16 bits) per cadascun en la forma que es mostra a continuació:

Bits 0-8	Ajustament (valor que caldrà sumar o restar al volum actual)
Bit 9	Signe (indica si cal sumar o restar l'ajustament)
Bits 10-12	Codi originador (per saber si el guany ha estat calculat automàticament, amb la intervenció de l'usuari, etc.)
Bits 13-15	Nom del codi (per saber si es tracta del *radio gain* o de l'*audiophile gain*)

Tota aquesta informació s'emmagatzemarà en el fitxer MP3 en forma *tags* seguint la següent proposta:

<Header for 'Replay Gain Adjustment', ID: "RGAD">		
Peak Amplitude		$xx $xx $xx $xx
Radio Replay Gain Adjustment		$xx $xx
Audiophile Replay Gain Adjustment		$xx $xx
Header consists of:		
Frame ID	$52 $47 $41 $44	= "RGAD"
Size	$00 $00 $00 $08	
Flags	$40 $00	(%01000000 %00000000)

Es tracta, en definitiva, d'un projecte ambiciós, que intenta portar a terme una bona idea, però al qual li falta molta ajuda per poder tirar endavant i ser considerat com un mètode vàlid a l'hora de normalitzar el volum de fitxers de so.

3. La proposta

3.1. Divisió del projecte

Degut a la dificultat que implica la realització de la totalitat del projecte, s'ha decidit dividir-lo en diverses parts.

3.1.1. Interpretació del format MP3

Es tracta de descodificar els components freqüencials d'un fitxer MP3, analitzar el fitxer seguint la definició de l'estàndard (iso11172-3)

A més, cal tornar a escriure el fitxer canviant el valor del guany global de cada *frame*. El valor a escriure el subministra el següent apartat.

3.1.2. Càlcul del factor de normalització

Utilitzant el resultat de la primera part de l'apartat anterior, es tracta de calcular el guany a aplicar. Per fer-ho, no n'hi haurà prou amb trobar el pic del fitxer, caldrà ponderar les freqüències depenent de la percepció que se'n tingui, realitzar una sèrie d'histogrames, etc.

3.1.3. Aplicació GUI

Es tracta de desenvolupar una interfície gràfica d'usuari que permeti aplicar la normalització a fitxers MP3. És una part de l'aplicació que té poc a veure amb els fitxers MP3, però és una part necessària per dotar al projecte d'un aspecte més presentable.

3.2. Interpretació del format MP3

3.2.1. Anàlisi

Com s'ha vist a l'apartat anterior, aquesta part del projecte es divideix en dues parts.

Primer de tot, cal analitzar el fitxer MP3 seguit les indicacions de l'estàndard. Per fer-ho, ens hem basat en el codi del descodificador MPEG que distribueix la pròpia ISO, i que es pot trobar, per exemple, a mp3-tech.

Un cop analitzat i descodificat el fitxer MP3, la part que ha de calcular el factor de normalització ja disposa de tota la informació que necessita per realitzar els seus càlculs i trobar el resultat que necessitarem per realitzar el següent pas.

El següent i darrer pas consisteix a generar el fitxer normalitzat. Per fer-ho, n'hi haurà prou amb copiar el fitxer original però canviant el guany global de cada *frame* (o quadre). Aquest nou guany global s'obté realitzant la següent operació:

nou_guany = guany_ant + 20 * log10 (gain)

On

nou_guany és el valor que guardarem al fitxer normalitzat

guany_ant és el valor del fitxer sense normalitzar

gain és el valor proporcionat per l'altra part del projecte (que pot ser negatiu si es tracta d'atenuar la senyal)

3.2.2. Mode d'utilització

Juntament amb aquest document es distribueix el codi font de l'aplicació en un fitxer anomenat **normalitzar.tgz**.

La instal·lació i posterior utilització de l'aplicació és molt senzilla, només cal seguir aquests passos:

1. Disposar d'un PC amb el sistema operatiu Linux.
2. Copiar el fitxer *normalitzar.tgz* al directori on es vulgui instal·lar l'aplicació.
3. Descomprimir el fitxer executant *tar xvfz normalitzar.tgz*
4. Entrar al directori *normalitzar* creat en el procés de descompressió.
5. Executar *./configure* i, posteriorment, *make*.
6. Amb això ja tenim l'executable creat. Ara només cal entrar al directori *normalitzar* i executar *./normalitzar nom_fitxer* (per facilitar les coses, l'aplicació disposa d'un fitxer d'exemple, anomenat *corrs.mp3*; així, només cal executar *./normalitzar corrs.mp3* i ja està)

El fitxer que volem normalitzar s'ha de passar com a paràmetre a l'hora de cridar el programa. Com a resultat, el programa generarà un fitxer MP3 normalitzat anomenat *nomfitxer_norm.mp3* (assumint que el fitxer que volíem normalitzar es diu *nomfitxer.mp3*).

El procés de normalització pot ser força llarg, depenent de la potència de l'ordinador on s'executi. Per exemple, amb una CPU a 350 MHz, el procés triga entre 10 i 15 minuts.

3.2.3. Codi font comentat

El codi font es distribueix entre el fitxer font (*main.c*) i el fitxer de capçalera (*normalitzar.h*).

El codi de *main.c* és el següent:

```
/***************************************************************************
                          main.c  -  description
                             -------------------
    begin                : mar jun 18 17:14:33 CEST 2002
    copyright            : (C) 2002 by David Pujadas
    email                : dpujadas@uoc.edu

***************************************************************************/

/***************************************************************************
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

***************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "normalitzar.h"


// Mentre no estigui implementada la part que calcula el nou guany,
// l'hem de silmular
float gain[2]={1.15, 1.15};

int calcFrameLen(struct MP3_header hdr)
{
  int bitrate, samprate;
```

```
  int * brvect;

  brvect = iBitRate[hdr.version != 3][hdr.layer];
  bitrate = brvect[hdr.bitrate];
  if (bitrate<0)
  {
    bitrate=0;
  }
  samprate = iSampleRate[hdr.version][hdr.freq];
  if (samprate<0)
  {
    samprate=0;
  }

  if (bitrate == 0 || samprate == 0)
  {
    return 0;
  }
  else
  {
    if (hdr.layer == 3)
    {
      return ((12000 * bitrate / iSampleRate[3][hdr.freq] +
hdr.padding) << 2)-4;
    }
    else
    {
      return (144000 * bitrate / iSampleRate[3][hdr.freq] +
hdr.padding)-4;
    }
  }
}

void MP3Info(char * nom_fitxer)
{
  Bit_stream_struc  bs;
  int sync, done=FALSE, stereo, Max_gr, error_protection,
crc_error_count;
  int total_error_count, clip;
  unsigned long frameBits, gotBits = 0, frameNum = 0, bitsPerSlot,
samplesPerFrame;
  layer info;
  PCM FAR *pcm_sample;
  frame_params fr_ps;
  III_side_info_t III_side_info;
  III_scalefac_t III_scalefac;

  pcm_sample = (PCM FAR *) mem_alloc((long) sizeof(PCM), "PCM Samp");
  fr_ps.header = &info;
  fr_ps.tab_num = -1;
  fr_ps.alloc = NULL;

      open_bit_stream_r(&bs, nom_fitxer, BUFFER_SIZE);
  while (!end_bs(&bs))
  {
    sync = seek_sync(&bs, SYNC_WORD, SYNC_WORD_LNGTH);
```

```
frameBits = sstell(&bs) - gotBits;
gotBits += frameBits;
if (!sync)
{
  done = TRUE;
  break;
}
decode_info(&bs, &fr_ps);
hdr_to_frps(&fr_ps);
stereo = fr_ps.stereo;
if(fr_ps.header->version == MPEG_PHASE2_LSF)
{
         Max_gr = 1;
}
else
{
         Max_gr = 2;
}
error_protection = info.error_protection;
crc_error_count = 0;
total_error_count = 0;
switch (info.lay)
{
  case 1:
  {
    printf("Només tractem Layer III\n");
    break;
  }
  case 2:
  {
    printf("Només tractem Layer III\n");
    break;
  }
  case 3:
  {
    int nSlots;
    int gr, ch, ss, sb, main_data_end, flush_main ;
        int  bytes_to_discard ;
        static int frame_start = 0;

    bitsPerSlot = 8;
    if(fr_ps.header->version == MPEG_PHASE2_LSF)
              samplesPerFrame = 576;
        else
              samplesPerFrame = 1152;
    III_get_side_info(&bs, &III_side_info, &fr_ps);
    nSlots = main_data_slots(fr_ps);
    for (; nSlots > 0; nSlots--)  /* read main data. */
      hputbuf((unsigned int) getbits(&bs,8), 8);
        main_data_end = hsstell() / 8; /*of privious frame*/
    if ( flush_main=(hsstell() % bitsPerSlot) )
    {
      hgetbits((int)(bitsPerSlot - flush_main));
              main_data_end ++;
        }
```

```
        bytes_to_discard = frame_start - main_data_end -
III_side_info.main_data_begin ;
        if( main_data_end > 4096 )
        {
          frame_start -= 4096;
          rewindNbytes( 4096 );
        }
        frame_start += main_data_slots(fr_ps);
        if (bytes_to_discard < 0)
        {
          printf("Frame %d incomplet.  Frame
descartat.\n",(int)frameNum - 1);
          break;
        }
        for (; bytes_to_discard > 0; bytes_to_discard--)
          hgetbits(8);
        clip = 0;
        for (gr=0;gr<Max_gr;gr++)
        {
          double lr[2][SBLIMIT][SSLIMIT],ro[2][SBLIMIT][SSLIMIT];
          for (ch=0; ch<stereo; ch++)
          {
            long int is[SBLIMIT][SSLIMIT];
            int part2_start;
            part2_start = hsstell();
            if(fr_ps.header->version != MPEG_PHASE2_LSF)
            {

III_get_scale_factors(&III_scalefac,&III_side_info,gr,ch,&fr_ps);
            }
            else
            {

III_get_LSF_scale_factors(&III_scalefac,&III_side_info,gr,ch,&fr_ps);
            }
            III_hufman_decode(is, &III_side_info, ch, gr,
part2_start,&fr_ps);
            III_dequantize_sample(is, ro[ch],
&III_scalefac,&(III_side_info.ch[ch].gr[gr]), ch, &fr_ps);
          }
          III_stereo(ro,lr,&III_scalefac,
&(III_side_info.ch[0].gr[gr]), &fr_ps);
          for (ch=0; ch<stereo; ch++)
          {
            double re[SBLIMIT][SSLIMIT];
            double hybridIn[SBLIMIT][SSLIMIT];
            double hybridOut[SBLIMIT][SSLIMIT];
            double polyPhaseIn[SBLIMIT];

            III_reorder (lr[ch],re,&(III_side_info.ch[ch].gr[gr]),
&fr_ps);
            III_antialias(re, hybridIn,
&(III_side_info.ch[ch].gr[gr]), &fr_ps);
            for (sb=0; sb<SBLIMIT; sb++)
            {
```

```
              III_hybrid(hybridIn[sb], hybridOut[sb], sb,
ch,&(III_side_info.ch[ch].gr[gr]), &fr_ps);
            }
            for (ss=0;ss<18;ss++)
              for (sb=0; sb<SBLIMIT; sb++)
                if ((ss%2) && (sb%2))
                  hybridOut[sb][ss] = -hybridOut[sb][ss];
            for (ss=0;ss<18;ss++)
            {
              for (sb=0; sb<SBLIMIT; sb++)
                polyPhaseIn[sb] = hybridOut[sb][ss];
              clip += SubBandSynthesis (polyPhaseIn,
ch,&((*pcm_sample)[ch][ss][0]));
            }
          }
        }
        /* En aquest punt tenim tota la informació descodificada.
            El mòdul encarregat de calcular el nou guany té a la seva
            disposició tota la informació que necessita */
        break;
      }
    }
  }
      close_bit_stream_r(&bs);
}

int amplify (char * mp3_input_filename, char * mp3_output_filename,
float gain[2])
{
  union
  {
    unsigned char buf[4*65536];
    struct
    {
      unsigned char tram_1[5];
      unsigned char g1[2];
      unsigned char tram_2[5];
      unsigned char g2[2];
      unsigned char tram_3[5];
      unsigned char g3[2];
      unsigned char tram_4[6];
      unsigned char g4[2];
      unsigned char tram_5[3];
    } side_info;
  } framebuf;
  union
  {
    unsigned char g1[2];
    struct
    {
      unsigned valor1 :7;
      unsigned x :1;
      unsigned y : 7;
      unsigned valor2 : 1;
    } bits;
  } gain1;
```

```
  union
  {
    unsigned char g2[2];
    struct
    {
      unsigned valor1 :4;
      unsigned x :4;
      unsigned y : 4;
      unsigned valor2 : 4;
    } bits;
  } gain2;
  union
  {
    unsigned char g3[2];
    struct
    {
      unsigned valor1 :1;
      unsigned x :7;
      unsigned y : 1;
      unsigned valor2 : 7;
    } bits;
  } gain3;
  union
  {
    unsigned char g4[2];
    struct
    {
      unsigned valor1 :6;
      unsigned x :2;
      unsigned y : 6;
      unsigned valor2 : 2;
    } bits;
  } gain4;
  unsigned char crc[2];
  FILE * fin, * fout;
  int ok=1, final=0, fl, guany1=0, guany2=0, guany3=0, guany4=0,
llegits;
  struct MP3_header mp3h;

  if ((fin=open(mp3_input_filename,O_RDONLY))==NULL)
  {
    printf ("Error a l'obrir el fitxer d'entrada
%s\n",mp3_input_filename);
    return 0;
  }

  if ((fout=fopen(mp3_output_filename,"wb"))==NULL)
  {
    printf ("Error a l'obrir el fitxer de sortida
%s\n",mp3_output_filename);
    return 0;
  }

  while (!final)
  {
    llegits=read(fin,&mp3h,4);
```

```
if (llegits!=4)
{
  final=1;
  break;
}
fwrite (&mp3h, 4, 1, fout);
fl=calcFrameLen(mp3h);
if (fl>4)
{
  // Sembla que el frame és vàlid
  if (!mp3h.protection)
  {
    llegits=read(fin, crc, 2);
    fwrite (&crc, 2, 1, fout);
    fl=fl-2;
  }
  llegits=read(fin,framebuf.buf,fl);
  if (llegits!=fl)
  {
    // Frame incomplet, el copiem igualment
    fwrite(&framebuf.buf, llegits, 1, fout);
    final=1;
    break;
  }
  // Guany gr 1 canal 1
  strcpy(gain1.g1, framebuf.side_info.g1);
  guany1=gain1.bits.valor1 << 1 | gain1.bits.valor2;
  guany1=guany1+(int)(20*log10(gain[0]));
  gain1.bits.valor1=guany1 >> 1;
  gain1.bits.valor2=guany1 & 0x01;
  memcpy(framebuf.side_info.g1, gain1.g1, 2);
  // Guany gr 1 canal 2
  strcpy(gain2.g2, framebuf.side_info.g2);
  guany2=gain2.bits.valor1 << 4 | gain2.bits.valor2;
  guany2=guany2+(int)(20*log10(gain[0]));
  gain2.bits.valor1=guany2 >> 4;
  gain2.bits.valor2=guany2 & 0x0f;
  memcpy(framebuf.side_info.g2, gain2.g2, 2);
  // Guany gr 2 canal 1
  strcpy(gain3.g3, framebuf.side_info.g3);
  guany3=gain3.bits.valor1 << 7 | gain3.bits.valor2;
  guany3=guany3+(int)(20*log10(gain[0]));
  gain3.bits.valor1=guany3 >> 7;
  gain3.bits.valor2=guany3 & 0x7f;
  memcpy(framebuf.side_info.g3, gain3.g3, 2);
  // Guany gr 2 canal 2
  strcpy(gain4.g4, framebuf.side_info.g4);
  guany4=gain4.bits.valor1 << 2 | gain4.bits.valor2;
  guany4=guany4+(int)(20*log10(gain[0]));
  gain4.bits.valor1=guany4 >> 2;
  gain4.bits.valor2=guany4 & 0x03;
  memcpy(framebuf.side_info.g4, gain4.g4, 2);
  // Guardem el resultat
  if (!mp3h.protection)
  {
```

```
      }
      fwrite (&framebuf, fl, 1, fout);
    }
    else
    {
      puts("Error de format");
      final=1;
      break;
    }
  }
  close(fin);
  fclose(fout);
  return ok;
}

int main(int argc, char *argv[])
{
  char nom_fout[100]="";

  // Comprovem que s'ha passat el fitxer a normalitzar com a paràmetre
  if (argc!=2)
  {
    puts ("Mètode d'utilització: normalitzar NOM_FITXER");
    return 0;
  }

  MP3Info(argv[1]);

  strncpy(nom_fout,argv[1],strlen(argv[1])-4);
  strcat(nom_fout,"_norm.mp3");

  if (!amplify(argv[1], nom_fout, gain))
  {
    printf("Error al normalitzar\n");
  }

  return EXIT_SUCCESS;
}
```

I el codi de *normalitzar.h* és el següent

```
/***************************************************************************
                          normalitzar.h  -  description
                             -------------------
    begin                : Wed Jun 19 2002
    copyright            : (C) 2002 by David Pujadas
    email                : dpujadas@uoc.edu

***************************************************************************/

/***************************************************************************
*
 *    This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *              (at     your     option)     any     later     version.
*
***************************************************************************/

#include "common.h"
#include "huffman.h"
#include "decoder.h"

typedef short PCM[2][SSLIMIT][SBLIMIT];

static const int iBitRate[2][4][16] = {
{  {-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2},
   {0, 32, 40, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320,
-1},
   {0, 32, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 384,
-1},
   {0, 32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416,
448, -1}
},
{  {-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2},
   {0, 8, 16, 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, -1},
   {0, 8, 16, 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, -1},
   {0, 32, 48, 56, 64, 80, 96, 112, 128, 144, 160, 176, 192, 224, 256,
-1}
} };

static const int iSampleRate[4][4] =
{  {11025, 12000, 8000, -1},
   {-2, -2, -2, -2},
   {22050, 24000, 16000, -1},
   {44100, 48000, 32000, -1}
};
```

```
struct MP3_header
{
  unsigned sync1       :8;
  unsigned protection  :1;
  unsigned layer       :2;
  unsigned version     :2;
  unsigned sync2       :3;
  unsigned private     :1;
  unsigned padding     :1;
  unsigned freq        :2;
  unsigned bitrate     :4;
  unsigned emphasis    :2;
  unsigned original    :1;
  unsigned copyright   :1;
  unsigned extension   :2;
  unsigned channel     :2;
};
```

3.3. Què queda per fer?

L'aplicació generada no està acabada ni molt menys.

A l'hora de generar el fitxer normalitzat, hem agafat una funció d'una altra aplicació per calcular la longitud dels *frames* (*calcFrameLen*). És una funció que funciona amb el 95% dels *frames*, però amb alguns dóna un resultat erroni que fa que el fitxer generat no es pugui reproduir. Per tant, una de les primeres coses a fer seria retocar aquesta funció perquè retornés un resultat correcte en el 100% dels casos.

Tenint en compte la gran quantitat de variants que por presentar un fitxer en format MP3 (*bit-rate*, freqüència de mostreig, modes, etc.), només hem tingut en compte el cas de fitxers generats en mode estèreo. Per tant, una forma de millorar la feina feta seria ampliar els formats suportats a tots els possibles.

Per altra banda, al final no s'ha realitzat la interfície gràfica proposada en un principi. Aquesta podria ser una altra forma d'acabar de perfilar una aplicació amb cara i ulls.

Una altra tasca interessant relacionada amb el projecte seria ampliar el nombre de formats de fitxer suportats, sense limitar-se a l'MP3 (Ogg Vorbis, AIFF, etc.).

I per acabar, un cop estigués finalitzat tot el projecte, amb tot el que queda per fer, es podria alliberar el codi amb llicència GPL per tal de posar-lo a la disposició de tot aquell que vulgui col·laborar en el seu desenvolupament o simplement estigui interessat en la normalització del volum de fitxers de so en format MP3.

4. Glossari

MPEG Acrònim de *Motion Picture Expert Group*, és un grup de treball creat per desenvolupar estàndards genèrics per la representació de gràfics en moviment, l'àudio associat i la seva combinació.

ISO Acrònim de *International Organization for Standardization*, és una organització internacional encarregada de vetllar per la creació, desenvolupament i manteniment d'estàndards.

MP3 Nom amb què es coneix l'estàndard MPEG-1 Layer III, encarregat de codificar àudio d'alta qualitat en molt poc espai.

Fraunhofer Institut alemany que va desenvolupar l'algoritme de generació dels fitxers MP3 i que actualment n'ostenta la patent.

Layer Terme anglès per denominar les capes en què es divideix l'estàndard MPEG-1 i que n'indica la seva complexitat i qualitat.

Frame Terme anglès que denomina els quadres o parts en què es divideix un fitxer MP3.

Bit-rate Terme anglès que denomina la quantitat de bits que es posen a disposició per representar certa informació.

Codec Acrònim de codificador/descodificador, és la part de maquinari o programari que serveix per codificar i descodificar un format de dades concret.

5. Bibliografia

Tota la informació necessària per realitzar aquest treball s'ha extret d'Internet, de les següents adreces:

- www.mpeg.org
- www.mp3-tech.org
- MP3 Overview
- Normalize
- Replay Gain
- ISO 11172-3