BOI 2.0: DISSENY D'UNA IMPLEMENTACIÓ DE JPEG 2000

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Ian Blanes Garcia i dirigit per Joan Serra Sagristà.

Bellaterra, Juny de 2007

El firmant, Joan Serra Sagristà, professor del Departament d'Enginyeria de la Informació i de les Comunicacions de la Universitat Autònoma de Barcelona

CERTIFICA:

Que la present memòria ha estat realitzada sota la seva direcció per Ian Blanes Garcia

Bellaterra, Juny de 2007

Firmat: Joan Serra Sagristà

Agraïments

Gràcies a la gent del GICI, especialment a en Joan Serra i a en Francesc Aulí. Gràcies a la Marta (espero poder-te cuidar tant bé com tu m'has cuidat aquestes últimes setmanes de tanta feina). Gràcies a tota aquella gent que amb petits detalls m'han ajudat també a realitzar aquest projecte.

Índex

Índex de figures

Capítol 1

Introducció

1.1 Què és JPEG 2000

El JPEG 2000 és un estàndard de compressió d'imatges que utilitza tècniques estat de l'art basades en la transformada wavelet. És un estàndard creat pel Joint Photographic Experts Group (JPEG) que té per objectiu substituir la vella norma de JPEG basada en la transformada discreta del cosinus. El JPEG 2000 està publicat com a estàndard internacional conjuntament per la ISO i la ITU-T (ISO/IEC 15444:2000). La ISO és una organització no governamental que agrupa les agències d'estandardització de 157 paisos i la ITU-T és la secció de telecomunicacions de l'agència d'estandardització de Nacions Unides.

Els principals avantatges d'aquest nou format són la millor capacitat de compressió, les múltiples possibilitats d'operar amb les dades comprimides (transmissió progressiva per diferents paràmetres, accés aleatori a les dades, etc) i el fet que es pugui comprimir amb i sense pèrdua amb el mateix mètode.

1.2 Què és BOI

BOI és la implementació de l'estàndard JPEG 2000 per part del Grup de Compressió Interactiva d'Imatges (GICI) del departament d'Enginyeria de la Informació i les Comunicacions (DEIC). La denominació BOI està inspirada en el Parc Nacional d'Aigües Tortes i Estany de Sant Maurici a la Vall de Boí (com és tradició en les implementacions de JPEG 2000 Kakadu i Jasper).

Actualment el grup ha implementat la primera versió de BOI, que suporta la majoria de les característiques descrites en l'estàndard, i que aconsegueix la manipulació correcta de la majoria dels arxius. Aquest ha estat un treball que s'ha dut a terme, per la seva gran complexitat, durant els darrers quatre anys .

1.3 Motivacions

El fet que es decidís fer una implementació era per permetre entendre, criticar i millorar les tecnologies usades en JPEG 2000. Es pot dir que BOI 1.0 ha complert aquesta funció satisfactoriament. La versió 2.0 intenta continuar en la mateixa direcció aprofitant l'experiència anterior.

1.4 Objectius

La nova versió de BOI té per objectiu desenvolupar una implementació que pugui arribar a tots els extrems de l'estàndard on la versió anterior no va arribar. Les principals limitacions sobre les característiques del BOI 1.0 apareixen perquè, per poder-les preveure, s'hagués hagut de realitzar primer la seva implementació. Així doncs és l'objectiu de BOI 2.0 poder realitzar una implementació de JPEG 2000 amb tota l'experiència aportada per la primera versió, fent especial èmfasi en evitar els imprevistos que ja es poden preveure.

Un altre punt important és intentar assolir un disseny que sigui flexible. En el sentit que es pugui modificar fàcilment cap a altres direccions, sense que això impliqui haver-lo realitzat de forma massa general. Per tant, això es traduirà molts cops en factoritzar força més que el que caldria en una implementació que tingués uns requisits de mantenibilitat més baixos.

I finalment un altre objectiu que es vol assolir és el de treballar amb algorismes de memòria limitada o $O(1)$ quan sigui possible. Aquest és un objectiu que realment canvia tot els processos i organització interna, ja que s'ha de dissenyar una implementació en forma de pipeline on el resultat de cada stage passa directament al següent.

1.5 Contingut de la Memòria

Òbviament, no ha estat l'objectiu d'aquest projecte millorar el treball que ha realitzat un equip durant quatre anys en un projecte de final de carrera d'únicament sis mesos. El que s'ha intentat és introduir un disseny que sintetitzi tots els coneixements ja adquirits de cara a que en etapes posteriors es pugui completar. Tot i així, realitzar el disseny d'una implementació de l'estat de l'art en compressió d'imatges i estàndard internacional és, igualment, un bon repte.

Així doncs, en aquesta memòria s'hi podrà trobar un disseny d'una implementació de JPEG 2000, especialment amb decisions de disseny molt raonades sobre com abordar diferents punts. En els punts més sensibles, s'ha intentat seguir un mètode més formal per tal d'arribar a la solució més convenient.

Com a guany personal he tingut l'oportunitat de consolidar els coneixements de compressió d'imatges i en especial del JPEG 2000 adquirits en els seminaris organitzats pel GICI. Així mateix, aquest projecte és una bona base per a una futura recerca que es pugui fer.

Capítol 2

L'estàndard JPEG 2000

En aquest capítol es presenta l'estàndard de JPEG 2000. Es farà especial èmfasi en l'organització de les diferents unitats de processament sense entrar en excessiu detall en l'operació interna de cadascuna d'elles. El que s'intentarà és exposar la funcionalitat i les interaccions ja que és el que cal per tal de realitzar el disseny.

2.1 Descripció

El JPEG 2000 és un sistema de compressió d'imatges. De les seves principals característiques no només destaca la bona qualitat de compressió, sinó força més que el fan apropiat per un gran ventall de situacions [14]:

Millor compressió a baixes qualitats Els sistemes anteriors de compressió d'imatges pateixen deficiències molt notables quan la compressió és substancial, que s'acaben traduint en contraintuïtius artefactes visuals (figura 2.6).

To continu o binivell Està adaptat tant a la compressió de dades obtingudes a la realitat, com a les generades per un ordinador. Les dades reals generalment tenen la característica que el to és continu en les diferents dimensions. En canvi, les generades per un ordinador contenen[14] zones de to similar unides per discontinuïtats (figura 2.1).

Amb i sense pèrdua El mateix procediment permet obtenir imatges amb o sense pèrdua de qualitat. Això es pot fer usant un mecanisme que permet la pèrdua gradual de qualitat d'una imatge ja comprimida (Post Compresion Rate/Distortion).

Transmissió progressiva La transmissió de les dades d'una imatge es pot fer de forma incremental, o sigui que amb només la transmissió parcial de les dades ja es pot obtenir algun resultat. És més, aquesta transmissió es fa enviant les dades més significatives primer, podent triar la significança de les dades segons la característica que interessi (e.g. resolució, qualitat, etc). A banda, la transmissió es pot fer de forma interactiva o a partir d'un fitxer estàtic.

Regions d'interès Es poden seleccionar zones de la imatge per tal que siguin comprimides amb una qualitat superior. Aquestes zones poden ser de forma irregular.

(a)

(b)

Figura 2.1: Exemple d'imatges en to continu (a) i to binivell (b)

Arquitectura oberta Permet estendre i ajustar-se a les necessitats de diferents usuaris. Existeix una base que tothom ha de complir i un seguit de possibilitats per estendre'l.

Robustesa a errors de bit El format de codificació de l'arxiu està especialment dissenyat perquè sigui resistent a errors, essent capaç de detectar l'error i re-sincronitzar les dades restants.

Seguretat El format d'arxiu preveu diverses mesures de seguretat com el xifrat, l'etiquetat o les marques d'aigua.

2.2 Parts

L'estàndard està dividit en múltiples parts, dividint la complexitat i delimitant les implementacions parcials d'aquest (figura 2.2). A data de la publicació d'aquesta memòria, totes les parts excepte la 7 i la 10 estan acabades.

Part 1 Core coding system (intended as royalty and license-fee free - NB NOT patent-free)

Part 2 Extensions (adds more features and sophistication to the core)

Part 3 Motion JPEG 2000

Part 4 Conformance

Part 5 Reference software (Java and C implementations are available)

Part 6 Compound image file format (document imaging, for pre-press and fax-like applications, etc.)

Part 7 has been abandoned

Part 8 JPSEC (security aspects)

Part 9 JPIP (interactive protocols and API)

Part 10 JP3D (volumetric imaging)

Part 11 JPWL (wireless applications)

Part 12 ISO Base Media File Format (common with MPEG-4)

Figura 2.2: Detall de les parts en les que està dividit l'estàndard JPEG 2000

2.3 Estàndard

És important tenir present que el que es defineix a l'estàndard és el procés de descompressió. És a dir, s'entendrà per JPEG 2000 tot allò que produeixi els resultats que s'indiquen a l'estàndard. Així es deixa oberta la porta a nous mètodes que reprodueixin els resultats anteriors, ja sigui més eficientment o més eficaçment.

Per tant, l'estàndard no és una bona referència sobre el procediment que s'ha de seguir per a la compressió. Tot i així, en els annexos de l'estàndard s'expliquen de forma bàsica mètodes senzills que defineixen per construcció parts dels resultats. Aquests ajuden a entendre de forma breu algunes de les unitats del procés.

De totes maneres, per entrar en detall en cadascuna de les etapes és molt recomanable documentar-se en articles o llibres que facin molt més èmfasi en el procediment. Així doncs, és fàcil trobar molta bibliografia dedicada a detallar, millorar els procediments, o proposar-ne de nous.

2.4 Procediment

El JPEG 2000 defineix una transformació des de l'espai comprimit fins a l'espai d'imatges. Aquesta transformació és reversible i està formada per múltiples etapes. Tot i que a l'estàndard el que es defineix és la transformació des de l'espai comprimit al descomprimit, s'entendrà com a transformació directa el fet de passar de l'espai descomprimit al comprimit i per la transformació inversa serà el contrari.

Comprimir una imatge és el fet de passar les dades per una sèrie d'etapes consecutives que acaben produint un arxiu comprimit (figura 2.3). Aquestes etapes són seqüencialment lògiques. És a dir, les dades han de passar per totes les etapes d'una en una en ordre. Tot i així, a l'hora de realitzar la implementació es pot manipular aquesta seqüencialitat de les operacions utilitzant estructures de processament més complexes (e.g. es pot usar pipeline per treballar amb memòria limitada i millorar l'eficiència de cache), sempre que es respectin les dependències lògiques de les dades.

El procediment consta de 10 grans etapes. S'expliquen en l'ordre de la transformada directa. Les dades que surten d'una etapa entren a la següent en bloc (pot ser en forma de flux en la majoria de les etapes, però per simplificar l'explicació de moment s'entendrà que no).

2.4.1 Imatges Raw

Una imatge raw (crua) és la forma més senzilla de representar una imatge. En les imatges en escala de grisos es tracta d'una matriu on a cada posició de la matriu li correspon el valor de la intensitat d'aquell punt. En les imatges en color es descomposa el color en els 3 components bàsics d'aquest (e.g. una descomposició podria ser separar-lo en els components R, G i B), i per cada component es defineix una matriu, com si d'una imatge en escala de grisos es tractés. També existeixen les imatges on hi ha més de 3 components, com per exemple, les imatges que representen l'espectre visible i a més l'espectre infraroig.

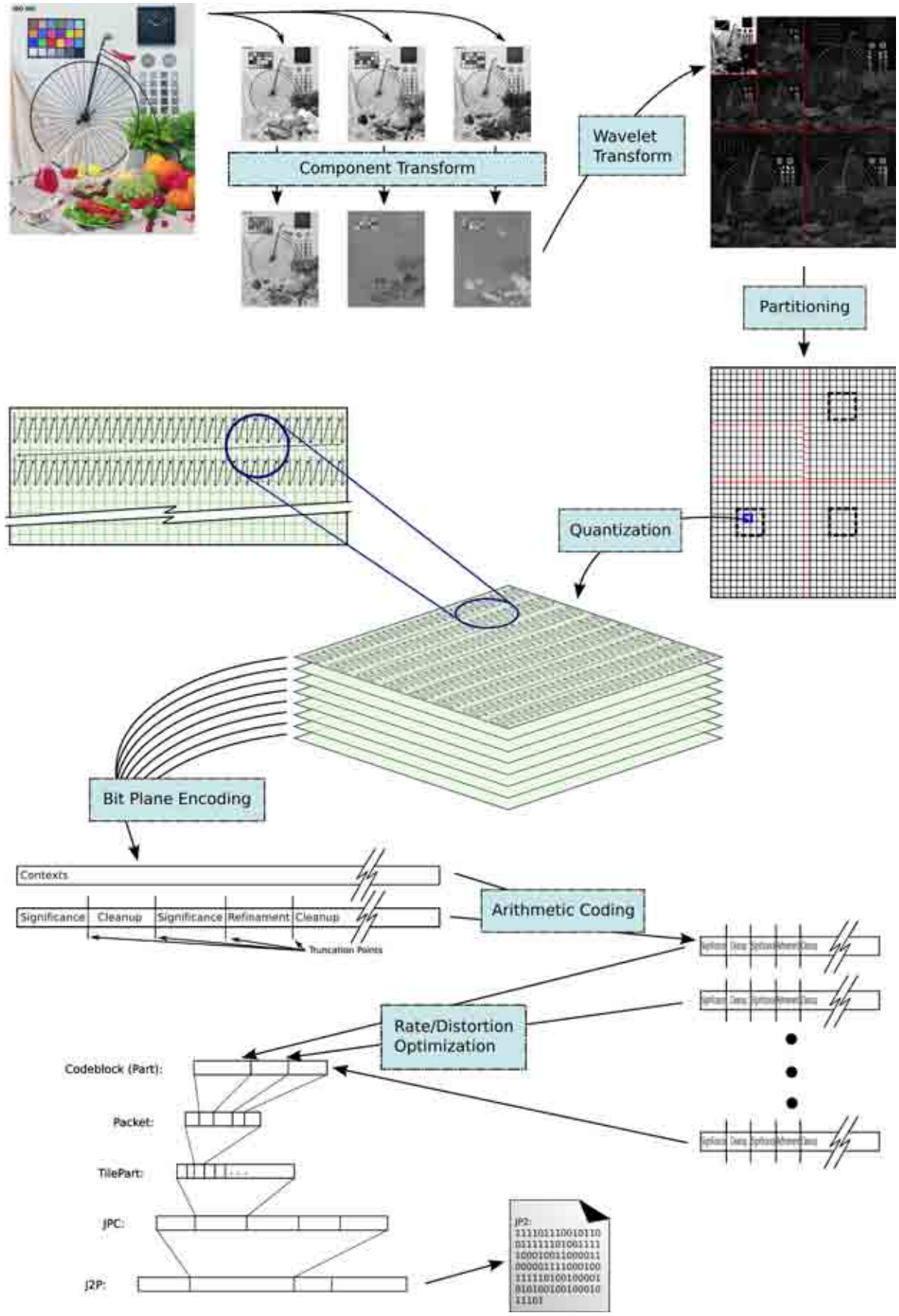


Figura 2.3: Esquema bàsic del flux de dades

Component 1 (vermell):

164	63	75	95	157
120	135	55	75	116
99	132	60	54	100
64	150	113	50	81
110	130	162	60	76

Component 2 (verd):

150	57	43	94	140
125	97	35	72	86
74	118	41	46	106
79	144	98	51	89
84	132	132	46	83

Component 3 (blau):

71	31	10	46	73
62	33	23	24	46
31	46	36	29	51
48	57	35	28	54
39	46	52	42	40

Figura 2.4: Exemple de les matrius per cada component de la sub-imatge de 5x5 píxels

2.4.2 Tiles

Els tiles (rajoles) són un conjunt de zones rectangulars de la imatge original que es divideixen. Aquesta divisió no permet tiles sobreposats, ni de diferents mides, ni deixar espais buits entre ells. De fet, aquesta etapa ve definida únicament per l'alçada i l'amplada del tile[1].

El propòsit d'aquesta etapa és únicament el de reduir la complexitat de la compressió. Hi ha etapes que depenent de la implementació requereixen memòria proporcional a la mida de la imatge per tant, limitant la mida de les imatges que es poden comprimir per la memòria del sistema on funcionen. Hi ha implementacions que usen tècniques per mitigar la relació entre la mida de la imatge i la memòria requerida, i per tant, eviten l'ús de tiles. Evitar l'ús de tiles és, en efecte, usar-ne un de la mida de la imatge.

La transformada wavelet (una etapa posterior al tiling), produeix el seu resultat en funció del context. Aquest és el principal motiu pel qual no es recomana usar tiles, ja que a les unions entre dos tiles hi apareixen artefactes força visibles a l'ull humà (figura 2.6).

Figura 2.5: Exemple de tiles.
Aquesta imatge de (512,512) píxels ha estat dividida en tiles de (150,150). Es pot observar que degut a que 512 no és múltiple de 150 hi ha tiles que no han quedat plens.

2.4.3 Level Shift

Es tracta d'una operació que s'aplica en el cas que s'estigui comprimint una imatge on els seus valors estiguin definits sense signe. Simplement el que es fa és que es canvia el rang dels valors des de $[0, 2n-1]$

[1] Per ser del tot correcte cal dir que també existeix un offset pel primer tile i un factor de conversió de mida de tile per cada component de la imatge. A aquesta relació entre els tiles dels diferents components se l'anomena grid.

Figura 2.6: Exemple d'artefacte visual.
En aquest cas es tracta d'un artefacte produït per la compressió basada en blocs de JPEG

fins a $[-n, n-1]$. L'única finalitat d'aquesta operació és la simplificació de les etapes posteriors a l'evitar que es produeixin overflows aritmètics que s'haurien de tractar de forma específica.

2.4.4 Transformada de components

La transformada de components és una operació que transforma l'espai dels components. Normalment es tracta d'una aplicació bijectiva $T : (x_1, \ldots, x_n) \mapsto (y_1, \ldots, y_m)$ on $n = m$, que s'aplica a cada coordenada de la imatge usant els n components d'aquesta com a entrada. Es pot donar el cas que $n \neq m$ o que per errors d'arrodoniment l'aplicació no sigui totalment reversible.

La transformada de components és una operació de decorrelació. És a dir, que redueix la correlació. Així doncs el que fa és deslligar la informació dels diferents components. Amb els components decorrelacionats s'aconsegueix una millor compressió. Aquesta és l'única fase que extreu la redundància entre els diferents components.

El level shift i la transformada de components es consideren també com una etapa de preprocessament i, moltes vegades, se'ls anomena així a la bibliografia. Cal tenir present que a partir d'aquesta etapa la compressió es realitza de forma independent per cada component. Per tant, en les futures etapes s'explicaran les operacions per a un sol component, podent estendre el procediment simplement realitzant la mateixa operació als altres.

A la Part 1 de l'estàndard, es defineixen dues transformades de components que totes les implementacions han de suportar (a part de la transformació identitat). Aquestes es presenten a continuació:

Reversible Color Transform

És una transformació lineal en la que s'ha fet especial èmfasi en que fos reversible per a tots els casos. Aquesta operació es pot fer servir per comprimir tant amb pèrdua com sense, però només pot ser usada amb la transformada wavelet reversible 5/3 que es veu més endavant.

Irreversible Color Transform

També és una transformació lineal, però que no és exactament reversible, tot i que produeix millors compressions que la Reversible Color Transform. Aquesta operació només es pot fer servir per comprimir amb pèrdua i només pot combinar-se amb la transformada wavelet irreversible 9/7.

2.4.5 Transformada Wavelet

Aquesta és potser una de les etapes més complexes de tot el procés. Tracta en separar successivament les altes freqüències de les baixes, o els detalls del general. Això es fa mitjançant l'ús de filtres passa altes i passa baixes aplicats a la imatge. Aquests filtres s'han seleccionat de forma que siguin separables (i.e. que es poden aplicar de forma commutativa a dimensions diferents), i el que es fa és aplicar-los de forma vertical i horitzontal a la imatge. Aleshores aquesta queda descomposada en quatre sub-bandes (anomenades LL, HL, LH, HH segons siguin el resultat d'aplicar un filtre passa baix(L)/alt(H) horitzontal i un filtre passa baix(L)/alt(H) vertical).

Figura 2.7: Transformada Wavelet 2D.
Es pot observar com a les respectives sub-bandes HL, LH i HH queden els detalls horitzonals, verticals o diagonals.

El procés s'aplica de forma recursiva sobre la sub-banda LL per obtenir millors resultats, fet que resulta en l'arbre de sub-bandes que caracteritza els mètodes de compressió basats en transformada wavelet.

Figura 2.8: Múltiples nivells de transformada Wavelet 2D.

Així doncs, la transformada wavelet implica per construcció la possibilitat de poder accedir a diferents mides d'una imatge. A més a més, el fet de poder-la aplicar de forma global a la imatge permet l'eliminació dels artefactes que es produeixen als extrems; com per exemple els que es produeixen per la transformada discreta del cosinus a JPEG (figura 2.6).

De la mateixa manera que s'han definit transformacions de components reversibles i irreversibles, quantizadors reversibles i irreversibles, existeix també una transformada reversible (la $(5,3)$ o LeGall) i la irreversible (la $(9,7)$ o Daubechies)[1]. I també tenen les mateixes propietats. La reversible funciona una mica pitjor però és totalment exacta i la irreversible funciona millor però no és exacta del tot.

El que es busca en aquesta etapa és agrupar l'energia de l'imatge de forma que més endavant es puguin seleccionar les zones que aporten més informació per fer compressió amb pèrdua. En general, l'energia queda agrupada a la zona de freqüències baixes.

El que es fa per saber quanta energia aporta cada zona de la imatge a la imatge original és usar el teorema de Parseval, que diu que l'energia es conserva en les transformacions ortonormals. La transformada Wavelet en general no és una operació ortogonal (requisit per ortonormal), però existeix la possibilitat d'escalar els coeficients per un valor constant per cada sub-banda ($L_2 - norm$) que fa que el teorema es pugui aproximar. Així doncs es pot avaluar la quantitat d'energia que aportarà cada zona de la imatge a la imatge reconstruïda.

Cal tenir present a l'hora d'implementar aquesta etapa que tot i que es poden implementar els filtres mitjançant una convolució, existeix un altre mecanisme anomenat lifting [2, 15]. Aquest mecanisme té les avantatges que redueix de forma significativa el nombre d'operacions a realitzar i que es pot aplicar sobre la mateixa entrada.

2.4.6 Quantització

La quantització és el procés que converteix els coeficients en punt flotant resultants de la transformada wavelet a valors enters. Això ho fa dividint en intervals els coeficients i assignant un valor enter a cada interval. Per quantitzar cal mirar en quin interval cau un coeficient i assignar-li el valor corresponent. La mida dels intervals s'anomena pas.

Un punt interessant és que no és aquí on es perd la majoria de la informació, com en altres processos de compressió amb pèrdua[2]. Una de les característiques del quantitzador és que els intervals estan enumerats consecutivament i la codificació d'aquests es fa en binari. Per tant, un coeficient quantitzat es pot representar com una serie de bits en els que a mesura que es van llegint es va refinant més en quin interval es troba. Això vol dir que eliminar el bits menys significatiu d'un coeficient quantitzat el que fa és unir els intervals de dos en dos, fet equivalent a seleccionar intervals més grans.

És cert però que depenent de la mida seleccionada es poden produir imprecisions al reconstruir la imatge original, però és com l'ús de la Irreversible Color Transform, que no és exacte però no és on s'elimina la informació redundant.

La quantització és una operació dependent de la sub-banda on s'aplica i per tant a sub-bandes diferents es poden aplicar passos diferents.

2.4.7 Dead Zone Quantizer

Aquest és el quantitzador estàndard que es defineix a la part 1, i que han d'incorporar totes les implementacions. Simplement es tracta d'un quantizador uniforme de pas Δ_b, excepte per l'interval $(-\Delta_b, \Delta_b)$ al voltant de 0, que es considera un sol interval (la zona morta).

[2]En el cas de comprimir sense perdua és pot usar un quantitzador de zona morta amb pas 1 que per coeficients enters és reversible

2.4.8 Codeblocks

Aquesta és una etapa totalment organitzativa, de divisió o partició lògica. Simplement el que es fa és dividir cadascuna de les sub-bandes en zones quadrades de mida potència de 2 i com a mínim 32 (e.g. $(32, 32), (64, 64), \ldots$). Aquestes zones anomenades codeblocks estan formades per matrius rectangulars d'enters. A partir d'aquesta etapa els codeblocks es codifiquen de forma independent. En les següents etapes és on més temps de cpu es consumeix, per tant és una avantatge poder processar els codeblocks de forma paral·lela.

2.4.9 Codificació de plans de bits

Els codeblocks s'examinen bit per bit en el codificador per plans de bits, i es van guardant els contextos en els que s'han trobat. Aquests contextos estan basats només en els bits que ja s'han examinat anteriorment per poder-los reconstruir també al descomprimir. Aquests contextos que es van generant per cada bit que s'examina serviran després per afinar les probabilitats del compressor aritmètic de la següent etapa.

L'ordre en que s'examinen tots els bits d'un codeblock és fixat i està orientat a examinar primer els bits més significatius de cada enter que forma un codeblock i anar refinant progressivament. Així doncs el que es fa és descomposar la matriu d'enters que és el codeblock per convertir-lo en n matrius de bits, on n és el nombre de bits que té cada enter. Aquestes matrius (bitplanes) s'examinen per l'ordre de significança del bit que representen.

Dins de cada matriu de bits l'ordre també és marcat i és el següent: els bits s'examinen per blocs de 4 files de dalt a baix del codeblock, i dins d'aquests blocs s'examinen primer els 4 bits de la primera columna, després els 4 de la segona i així fins l'última.

En realitat, els bits s'examinen dins de cada bitplane tres vegades en l'ordre descrit anteriorment (els bitplanes es segueixen examinant només un cop en l'ordre descrit al començament). Però només es codifiquen en una de les passades.

Significance propagation Es diu que una posició d'un bitplane és significant si és 1. En aquesta fase es codifiquen el bits que tenen un dels seus 8 veïns directes significants. Un cop una posició de bitplane ha esdevingut significant, aquesta fase salta aquella posició en els bitplanes següents.

Magnitude refinement En aquesta fase es codifiquen els bits dels coeficients que han esdevingut significants en etapes de Significance propagation anteriors (excepte la immediatament anterior).

Clean-up Tots els bits del bitplane que no s'han codificat en etapes anteriors es codifiquen ara. Aquesta etapa incorpora un mode de codificació de run-length que permet codificar seqüències de coeficients insignificants.

A l'hora de generar la codificació del bit que s'examina, s'utilitza el context conegut, i a més a més es guarda per tal que a la següent etapa es pugui predir millor la probabilitat d'aquell símbol. El context està format pels valors dels 8 píxels adjacents i es simplifica agrupant diferents combinacions de píxels veïns ens contextos iguals.

D'aquesta etapa surt una seqüència de bits (bitstream) amb les codificacions de cada bit i una seqüència de contextos per cada bit del bitstream. A més a més també és produeix una llista amb les posicions on acaba i comença un nou bitplane, llocs anomenats punts de trencament.

2.4.10 Codificació per entropia

Els bitstreams resultants de la fase anterior es codifiquen en un codificador aritmètic MQ. El codificador aritmètic MQ és un codificador en el que es representa la codificació com un valor dins d'un interval. Partint d'un rang de valors donat, es divideix aquest rang en intervals proporcionals a la probabilitat d'aparició d'un símbol i es selecciona com a nou rang l'interval del símbol aparegut. Aquest interval es va refinant de forma successiva a mesura que van apareixent nous símbols. Un cop processats tots els símbols es tria un valor de l'interval resultant i es diu que aquest valor és la codificació aritmètica de l'entrada.

El codificador MQ treballa amb els contextos que s'han anat generant a la fase anterior per tal d'estimar millor els rangs de probabilitats. Aquesta estimació la fa a partir d'un autòmat finit que té per transicions els contextos que es van rebent.

Existeix també un mode mandrós en el que a partir del quart bitplane es deixen de codificar les dues últimes etapes de la codificació de plans de bits, aquesta és una optimització que té un impacte mínim en el factor de compressió però que millora substancialment la velocitat de la compressió per software.

Els punts de trencament proporcionats per la fase anterior, es propaguen a la fase següent i s'utilitzaran en la fase d'optimització del rate/distortion. La propagació es fa tenint en compte les noves mides que hi ha després del codificador aritmètic.

2.4.11 Layers, Paquets i Precincts

A l'arxiu resultat la informació està organitzada en paquets. Aquests paquets van aportant successivament informació sobre els codeblocks. Però ho fan d'un codeblock o un altre en funció de la característica que es vulgui prioritzar. De fet, el que es fa és establir un ordre sobre les característiques en funció de les quals es pot ordenar i refinar l'ordenació de la primera categoria amb la segona, etc.

Hi ha quatre característiques per les que està permès ordenar; són les següents: Layer, Resolució, Component i Posició. La característica que potser no queda clar a què es refereix és Layer, que simplement significa que es produeix un increment en la qualitat. Tot i existir $4! = 24$ ordenacions possibles amb 4 característiques, la part 1 de l'estàndard només permet les 5 següents ordenacions segons la seva possible aplicació:

Layer-Resolution-Component-Position (LRCP) Examinar imatges ràpidament i de forma progressiva millorant la qualitat d'aquelles que tinguin interès.

Resolution-Layer-Component-Position (RLCP) Quan calgui disposar d'una imatge en múltiples resolucions, per exemple per a dispositius de representació de diferents mides.

Resolution-Position-Component-Layer (RPCL) Un entorn on calgui llegir la imatge a múltiples resolucions i a més interessi accedir de forma parcial a zones de cada resolució, com per exemple en un mipmap.

Position-Component-Resolution-Layer (PCRL) Quan calgui accedir per zones a la imatge però sempre amb una resolució igual.

Component-Position-Resolution-Layer (CPRL) Quan cal disposar dels components de forma independent.

Aquestes cinc ordenacions són flexibles i enmig d'un arxiu es pot decidir canviar d'ordenació pels paquets que falten.

Els precincts s'utilitzen per agrupar els codeblocks en les sub-bandes altes. Això es fa perquè no és útil poder accedir a un codeblock de la subband HL si no s'accedeix també a l'equivalent de la sub-banda LH i HH (i aleshores aquesta informació es pot usar per refinar la resolució anterior). El precincts, a part d'ocupar tres sub-bandes diferents també poden contenir més d'un codeblock sempre que siguin el mateix nombre de codeblocks per a cada sub-banda, aquests siguin adjacents i la forma del precinct a cada sub-banda sigui un rectangle de les mateixes proporcions.

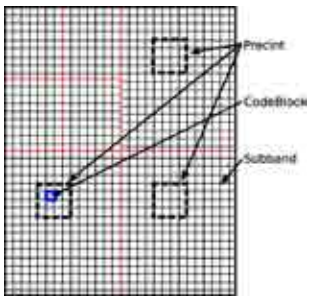


Figura 2.9: Sub-banda, precinct i codeblock

2.4.12 Optimització Rate/Distortion

Gràcies a que s'ha pogut propagar la quantitat d'energia que aporten els bitstreams en els seus diferents punts de possible trencament, es poden anar seleccionant ordenadament trossos de bitstream i saber l'energia que aporten cadascun.

És el moment en el que es decideix separar la informació rellevant que cal preservar i descartar la resta. Això es pot fer ja que es disposa de tots els bitstreams degudament etiquetats amb els valors d'energia que aporten i la quantitat d'informació que requereixen per fer-ho (Rate/Distortion). Per tant es triaran els bitstreams que més energia aportin consumint menys informació.

Una vegada determinat un ordre de progressió a optimitzar es van seleccionant progressivament les parts dels bitstreams que milloren més l'objectiu a optimitzar guiant-se pels valors de rate/distortion en els punts de trencament etiquetats.

2.4.13 Organització de l'arxiu

Finalment, una vegada es té la informació que s'haurà de guardar, s'encapsula en un format d'arxiu ".jpc" (jpeg codestream). Aquest arxiu ".jpc" conté la informació necessària per poder comprimir i descomprimir el seu contingut, i a l'hora pot ser encapsulat també en un arxiu ".jp2" que afegeix informació addicional sobre com ha de ser visualitzat el contingut (i.e. espais de color) i altres metadades.

Aquest encapuslat es fa insertant encapçalaments als diferents paquets per tal de poder accedir després a l'arxiu de forma no lineal sense haver de descomprimir-lo tot. Això permet realitzar moltes operacions en un arxiu comprimit, com per exemple reordenar les seves dades o seleccionar-ne zones.

La creació dels encapçalaments i organització de l'arxiu s'anomena Tier-2, i la compressió aritmètica de les dades vista fins les etapes anteriors Tier-1. El Tier-2 és la capa més externa d'un arxiu, que cal descodificar per poder-lo indexar.

Per tal que els encapçalaments de cada paquet no ocupin un espai excessiu, es comprimeixen. Aquesta compressió es fa codificant la informació de forma eficient, per exemple s'usa el primer bit d'un header per indicar si el paquet es buit o es codifiquen les aparicions d'un codeblock en forma de tag-tree (un tag-tree és una estructura que és capaç d'explotar la redundància espacial similar a la presentada a [13]).

2.5 Resum

Fins ara s'han presentat les diferents etapes del procés de compressió JPEG 2000. Amb aquesta explicació s'ha intentat aconseguir una bona visió global del flux de dades i de les principals característiques de cada etapa.

Aquesta explicació serveix de base de coneixements per poder entendre els conceptes i els raonaments de les etapes posteriors.

Capítol 3

Anàlisi dels requisits

En aquest capítol es detallen els requisits que es van establir per aquest disseny. El requisit principal és que la implementació s'ajusti a l'estàndard JPEG 2000 i tot seguit es requereixen detalls ja sigui concrecions on l'estàndard deixa elecció o sobre com abordar els problemes basats en l'experiència anterior del GICI. També hi ha alguns requisits pel que fa a les tecnologies en les que es recolzarà la implementació (clarament continuïstes en relació a la versió anterior).

Cal tenir en compte que el primer requisit funcional fa referència a l'estàndard ISO/IEC 15444-1:2000 [6]. Aquest per si sol està format per 218 pàgines de requisits, que tot i que seria redundant enumerar a continuació, sí que s'han de tenir presents.

3.1 Funcionals

Llibreria de compressió / descompressió de jpeg2k La llibreria ha de poder gestionar tot el procés de compressió i descompressió per l'estàndard ISO/IEC 15444-1:2000 (part 1) [6] i preveure l'ampliació per les parts 2 [7], 3 [8] i 9 [9].

Formats d'arxiu S'ha de preveure suport pels següents formats d'arxius en els que es poden encapsular els codestreams de jpeg2k: jpc, jp2, jpx, mj2 (motion jpeg2k). Això no inclou suportar els diversos processos de compressió que caldrien per arribar-hi (específicament el motion jpeg2k).

Progression orders S'ha de permetre flexibilitat a l'hora de triar els diferents progression orders dels arxius.

Grid S'ha de contemplar el model de graella a l'hora d'especificar tiles en diferents components.

Tipus de dades S'han de suportar operacions amb diferents tipus de dades amb l'ús de templates on sigui possible. S'han de tenir en consideració els possibles efectes al triar els diferents tipus de dades.

Extractor S'ha de poder extreure zones de la imatge de forma arbitrària en el descompressor.

Carregar arxius PGM/RAW S'ha de disposar d'un front end que permeti la lectura de fitxers PGM i RAW.

Parser de parametres avançat S'han de poder parsejar tots els paràmetres que requereixi l'aplicació. S'ha de dissenyar un parser extensible i modular.

3.2 No funcionals

3.2.1 Requisits de disseny

Director La gestió del procés ha d'estar centralitzada en un director de procés que gestioni el flux d'informació a les diferents etapes. Aquest director ha de ser substituïble segons la funció que es requereixi per la llibreria. Els diferents rols que ha de poder portar a terme el director són: compress, decompress, extract i eye.

Estructura de dades centralitzada Les dades han d'estar centralitzades, per poder aplicar optimitzacions globals en el cas que calgui.

Escalable Els algorismes usats han de tenir complexitats acceptables, o sigui no exponencials.

Memòria limitada La mida de les imatges que ha de ser capaç de comprimir, no ha d'estar limitat per la memòria disponible en el sistema.

Paral·lelitzable S'ha de possibilitar la compressió en paral·lel de diferents parts del sistema.

Reusable, Mantenible i Extensible S'ha de dissenyar per tal que el codi sigui fàcilment reusable, mantenible i extensible.

Parser capaç de separar els mòduls El gestor de la configuració ha de ser capaç de funcionar amb part dels mòduls del sistema.

3.2.2 Requisits de fiabilitat

S'haurà de preveure que les diferents parts s'hauran de poder testejar amb les eines JUnit i Coverlipse. En la creació dels JUnits es poden usar parts de BOI 1 per comprovar els resultats. També es pot efectuar el procés de forma inversa i comprovar l'entrada. S'ha de procurar que es pugui usar el software Kakadu[16] per tal de validar la compressió/descompressió.

3.2.3 Requisits de portabilitat

La llibreria ha de ser compatible amb les diferents plataformes utilitzades avui en dia. Això es traduirà en que funcioni correctament en les plataformes on hi ha una versió mantinguda de Java del fabricant Sun Microsystems. A efectes de portabilitat només es requereix que funcioni la versió en bytecode. Compilació Binari/bytecode. S'ha de preveure que la llibreria s'haurà de poder compilar tant a bytecode java com a codi màquina. El bytecode es generarà amb el compilador de Sun Microsystems i el binari amb la versió de Java del GNU Compiler Collection.

3.2.4 Requisits legals

Es requerirà que les llibreries de les que depengui la llibreria que s'està dissenyant no imposin restriccions a la resta de codi. Es preferiran les llibreries amb llicències LGPL o similars. Per tant, no es farà servir la llibreria Java Advanced Imaging[1].

3.2.5 Requisits de seguretat / privacitat

Es procurarà que no es puguin produir problemes de seguretat a l'hora de llegir imatges especialment manipulades per tal de causar problemes a la llibreria. Tot i que s'entén que usant el llenguatge de programació Java això és molt difícil.

3.3 Resum

Aquests requisits apunten en tres grans direccions el disseny que es demana:

- Realitzar un disseny per a una implementació de JPEG 2000. Aquest és un requisit simple d'expressar però molt complex en realitat, tot i que no per ser tant complex és menys comprobable.
- JPEG 2000 és una tecnologia que pertany a l'estat de l'art en compressió d'imatges. Això implica que cal tenir present que és molt susceptible de variar en direccions que no són clares. Per tant cal fer un disseny que permeti una alta mantenibilitat i extensibilitat. I el que és més difícil és fer-ho i obtenir un disseny que no perdi competitivitat per flexibilitat. Aquesta flexibilitat s'ha d'enfocar generalitzant quan el cost d'aquesta operació no sigui excessiu, però sobretot intentant minimitzar el cost d'adaptació a novetats. Per tant ja es comença a entreveure que la factorització serà un detall clarament a tenir en compte en el següent capítol.
- Aquest disseny és una segona versió i per tant, perquè tingui èxit cal no ensopegar dues vegades amb la mateixa pedra. Això s'acabarà traduint en moltes consultes amb les persones que van realitzar la primera versió, detectar-les, analitzar les problemàtiques i intentar evitar-les o mitigar-les. Aquestes problemàtiques estan expressades anteriorment en forma de requisit, resta per tant, analitzar-les i solucionar-les en la mesura del possible.

[1] Sun Microsystems, propietari de la llibreria, ha manifestat públicament la seva intenció d'alliberar l'API de java en GPL. Però, és conegut que está tenint dificultats, sobretot en les llibreries gráfiques, degut a que moltes d'aquestes parts són rellicenciades d'altres companyies

Capítol 4

Disseny

En aquest capítol explicaré el disseny que es proposa per aquest projecte de final de carrera com a base per BOI 2.0.

Com a mètode per arribar als dissenys de classe que es presenten a continuació el que s'ha fet és primerament fer la divisió en mòduls del projecte: Estructura de dades, Recursos de càlcul, Directors i Gestor de paràmetres. Després per cadascuna d'aquestes parts s'ha raonat sobre la interacció que hauria de poder tenir amb l'exterior, sempre tenint present que aquesta no s'hauria de restringir al context actual i s'haurien de dissenyar de la forma més general que es pugui. Per tant en comptes de determinar primer les interaccions actuals i després dissenyar interfícies que les complissin, el que s'ha fet és dissenyar primer les interfícies i després veure que complien totes les interaccions específiques.

Per fer això s'han anat plantejant un seguit de qüestions de disseny sobre els diferents punts, que s'han anat valorant de forma individual i argumentant en cada cas per decidir quina opció triar. També és cert que per realitzar el disseny de l'estructura de dades s'ha seguit una metodologia més formal de cara a assegurar la correcció d'aquesta, ja que es tracta d'un punt important. Un cop realitzades les argumentacions necessàries s'ha procedit a fer els diagrames corresponents per a cada situació concreta, detallant més o menys en funció de la necessitat i les restriccions de temps.

A continuació es presenten les diferents etapes del projecte. S'ha intentat seguir l'ordre deductiu a l'hora de traspassar les argumentacions a la memòria. Primer s'han resolt les qüestions generals que podrien afectar a la resta del disseny i a trobar una organització bàsica de les diferents parts que s'adeqüés al problema a resoldre. Després s'ha procedit amb l'organització de les dades, creant una estructura per a elles sobre la qual un director global pugui actuar usant recursos de càlcul estancs. Els recursos de càlcul s'han definit de forma estanca en el cas que fos possible, sense entrar a descriure'ls en profunditat i definint les interfícies que s'adequarien a la línia general del disseny. Hi ha hagut casos com en el de la transformada wavelet, que s'ha hagut de fer un disseny més extensiu, per poder suportar característiques que no té el recurs de càlcul equivalent a BOI 1.0. Per gestionar el flux de control es fan servir directors, que poden tenir funcionalitats compartides i per tant, s'han dissenyat de forma que es puguin reutilitzar. També ha calgut dissenyar un gestor de paràmetres d'entrada que pogués servir per parsejar els paràmetres de diferents directors, essent l'objectiu d'aquest disseny minimitzar la complexitat exterior del gestor de paràmetres (incrementant potser la interior).

4.1 Qüestions generals

Dues qüestions generals que ha calgut resoldre a l'hora de decidir com dissenyar aquest projecte es presenten a continuació. La primera és estudiar l'ús dels mecanismes que proporciona el llenguatge de cara a simplificar la programació i veure si són viables d'aplicar en aquest cas concret. Per tant, s'ha hagut d'estudiar en quins àmbits es podria aplicar i determinar l'overhead que això comporta. La segona qüestió era la precisió dels tipus de dades; s'ha hagut d'estudiar sobre la penalització d'implementar operacions amb tipus de dades de doble precisió. La doble precisió és necessària per suportar imatges d'alta profunditat de bit.

4.1.1 Boxing i genèrics

L'ús de genèrics (els templates del java) es força interessant i ha estat una opció que s'ha considerat. Per exemple l'ús de genèrics permetria fer coses com les que es mostren a la figura 4.1, on totes les operacions puntuals deriven d'una mateixa classe base, que generalitza l'aplicació de les operacions puntuals als arrays. Així per definir una imatge que s'hagués d'aplicar a tot un vector, només caldria definir-la puntualment, com el "map" dels llenguatges funcionals. El problema rau en que en java els genèrics no creen un nou tipus de dades a l'inicialitzar-los, simplement el que fan és automatitzar casts a Object en temps de compilació. Per això el següent codi no és correcte:

```
class TestSumaGenerics<A,B> {
        A operacio(A parametre1, B parametre2) {
                return A + B;
        }
}
```

Com que A i B són del tipus Object no tenen la propietat de la suma. A més a més els genèrics no es poden inicialitzar amb tipus unboxed com int o float. Els tipus "unboxed" són excepcions al llenguatge pensades per optimitzar la velocitat d'execució. Aquests no hereten de la base comuna que es Object per totes les altres classes, de fet no són classes. També existeixen els tipus Integer i Float que són envoltoris als anteriors. Aquests són boxed, és a dir, són classes i per tant, es poden fer servir en els genèrics, però tenen força penalització a l'utilitzar-los. Per tal d'obtenir una idea quantitativa del cost de fer servir boxing, s'ha fet un petit benchmark, mostrat a la taula 4.1. S'ha vist que el cost és massa gran i per tant, s'ha decidit no usar boxing i tampoc usar genèrics en totes les situacions que requerissin tipus boxed.

Conclusions:

- És tant ràpid float com double en totalment unboxed però no en boxed
- Amb acumuladors unboxed hi ha un 10% de penalització en tipus
- La penalització rau principalment en els acumuladors boxed (tornar a fer les caixes)
- La penalització per boxing es del 50% si es fa bé
- La penalització per boxing mal feta és del 350%

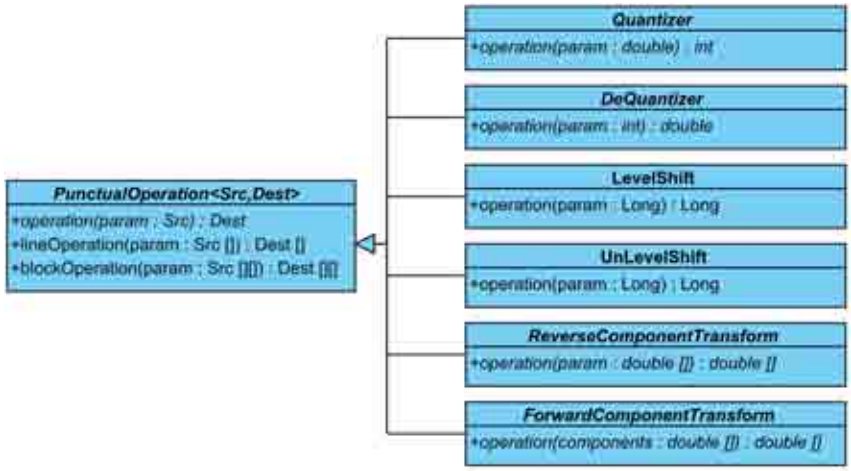


Figura 4.1: Exemple de genèrics, on totes les operacions puntuals es podrien derivar de la mateixa classe

Acumulador	Items	Run1(ms)	Run2	Run3	Avg	Comparacio tipus	
Unboxed double	Boxed double	125	109	125	119.67	1.09	9%
Boxed double	Boxed double	344	328	343	338.33	1.28	28%
Unboxed double	Unboxed double	78	78	78	78	1	0%
Boxed double	Unboxed double	297	281	282	286.67	1.04	4%
Unboxed float	Boxed float	109	110	109	109.33	0.91	-9%
Boxed float	Boxed float	265	265	266	265.33	0.78	-22%
Unboxed float	Unboxed float	78	79	78	78.33	1	0%
Boxed float	Unboxed float	281	281	265	275.67	0.96	-4%

Taula 4.1: Resultats del test de comparació entre boxed / unboxed i presició simple / doble

4.1.2 Precisió

Per tal de determinar la viabilitat d'implementar operacions amb dades d'altes precisions sense perjudicar la resta, s'ha estudiat la penalització que genera sobre el temps d'execució.

Latència: Nombre de cicles que calen per acabar d'executar una instrucció completament.

Throughput: Nombre de cicles que calen esperar abans de poder executar una altra instrucció del mateix tipus (sense que tinguin dependències de dades).

Totes les operacions en punt flotant tenen la mateixa latència i throughput independentment de la precisió que tinguin les dades [5]. Exceptuant la divisió que té una penalització de fins al 65% en la latència i el throughput si s'efectua en doble precisió en comptes de simple.

En la multiplicació de punt fix la penalització és del 300% en la latència i el throughput (appendix A).

4.2 Organització general

Requisits de disseny: Les dades s'organitzen en una sola estructura que permet accedir a totes i es gestionen a través d'un director omnipresent.

A continuació es presenta l'organització general que s'ha trobat millor:

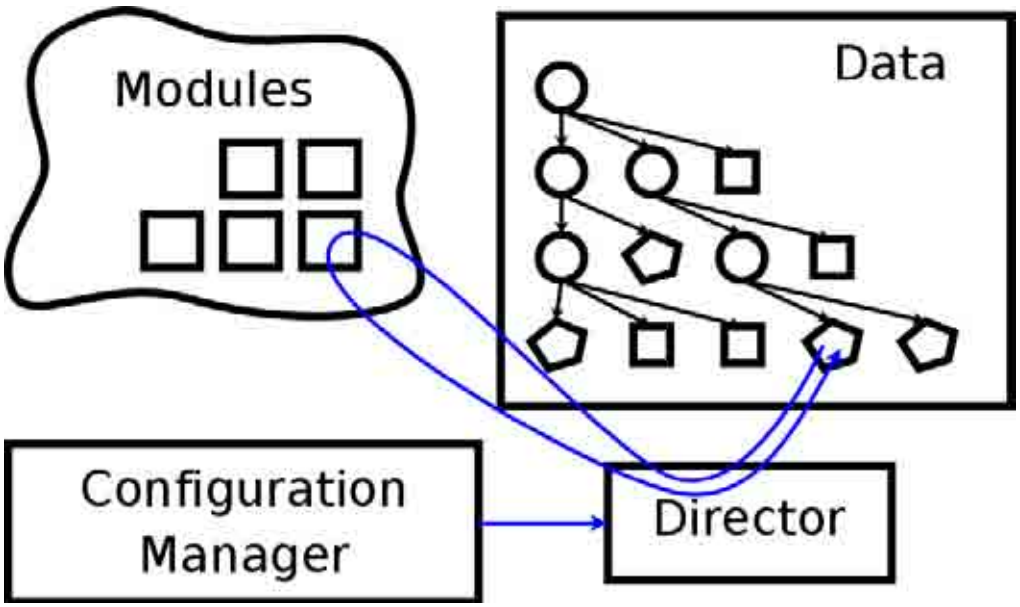


Figura 4.2: Organització general

Mòduls Els recursos de càlcul, tals com un codificador aritmètic MQ o una transformada Wavelet, es troben agrupats en aquest gran bloc. Cadascun d'ells funciona de forma independent, tant del flux de dades com dels altres mòduls. Això permet una agilitat substancial per maniobrar amb diferents configuracions d'aquests recursos que es puguin requerir. Simplificant també la depuració ja que aquestes unitats es poden comprovar de forma independent i un cop madures ja no caldrà mantenir-les gaire.

Dades Les dades es troben agrupades i indexades en una gran estructura. Això permetrà aplicar operacions globals i també facilitarà fer variacions al flux de dades quan sigui necessari.

Director En forma de gestor global, aquesta és la part que tot i que no té gaire complexitat organitzativa més s'espera que canviï. Per tant, és molt important que les altres parts estiguin ben aïllades dels canvis que aquí es produeixin.

Gestor de paràmetres El gestor de paràmetres actua de forma independent, fet que permet simplificar-lo i que maduri encara que es vagin fent canvis al director.

4.2.1 Motivació

- L'organització global permet arribar a solucions òptimes per camins sense sub-camins òptims.
- El fet que els recursos de càlcul siguin independents permet reutilitzar-los.
- És bo separar les dades a processar de les dades de control.
- El gestor de configuracions separat del director permet reaprofitar el codi del gestor i els arxius de configuració amb diferents directors.

4.2.2 Problemes

- Es complica el processament paral·lel, ja que ara és el director que ho ha d'organitzar i no es poden connectar directament les sortides d'uns mòduls amb les entrades d'uns altres.

4.3 Estructura de l'arbre

A continuació es presenten les decisions que han determinat la natura de l'estructura que organitza i centralitza totes les dades:

Requisit de disseny: Les dades s'han d'organitzar en forma d'arbre. Aquesta estructura és centralitzada i conté totes les etapes del procés de codificació (tot i que no cal que sigui al mateix temps).

Requisit de disseny: Els paràmetres de les dades s'han d'incloure a l'estructura d'arbre. Això permet una descripció centralitzada que pot gestionar diversos arbres alhora.

Decisió: S'han de codificar els paràmetres de les dades en dur, és a dir, una variable per paràmetre, a l'arbre?
Arguments:

- Així s'aprofiten les comprovacions del compilador.
- Els paràmetres són coneguts en temps de compilació.
- No provoca un excés de feina fer-ho (com és el cas dels paràmetres de configuració).
- També permet centralitzar la documentació dels paràmetres en forma de javadoc.

Decisió: Sobre si s'han d'usar a) tipus durs als nodes de l'arbre o b) tipus generals (e.g. Object) i anar fent casts.
Arguments:

- a) avança la comprovació de tipus en temps de compilació, en canvi b) ho deixa fins a l'execució.
- b) evita muntar un sistema d'herències per assignar diferents tipus a un mateix node.
- Amb b) s'han de fer casts tota l'estona.
- El projecte és força complex i requereix un bon disseny previ, pel que s'esperen pocs canvis de l'estructura de dades.
- Es poden usar templates per unificar tipus en un sol lloc.

Així doncs l'opció a) és més raonable.

Decisió: S'han d'afegir operacions de gestió als nodes de l'arbre.
Arguments:

- Si les operacions de gestió fan les transformacions sobre les dades, aleshores el director acabarà cridant una funció de l'arrel de l'arbre que farà tota la funcionalitat. Això fa que el director perdi el seu sentit.
- Es poden afegir operacions de gestió sobre l'estructura però no el contingut (e.g. Funcions de gestió dels sub-nodes: afegir, treure, ...).

Tot això s'ha acabat traduint per si s'ha de trencar l'encapsulació dels nodes de l'arbre. L'encapsulació seria útil si els objectes tinguessin mètodes (sinó òbviament no serveix perquè no es poden modificar les dades encapsulades).

Decisió: Es podrien separar els nodes del arbre en dos tipus: els estructurals i els de dades. En els primers es podrien usar objectes de les llibreries java (List, etc), i en els segons objectes propis.
Arguments:

- Usar objectes estructurals de java simplifica força la gestió i permet l'ús de syntaxs específiques per la seva manipulació sense gaire feina addicional (i.e. Foreach).
- En els nodes de dades es poden fer servir getters i setters per encapsular l'estructura interna, però al final acabaran encapsulant simplement un array o un valor, cosa que produirà un overhead de programació força gran.

Decisió: Emmagatzemar les dades de la imatge original per línies.
Arguments:

- D'aquesta manera es poden eliminar fàcilment dades de la imatge original de la memòria.
- Si cal es poden generar les simetries als extrems de dalt i baix de la imatge amb dos referències a la mateixa línia per cada línia simètrica.

Decisió: Retenir els coeficients resultats de la transformada wavelet a aquest nivell i fer servir referències als nivells més profunds.
Arguments:

- Els següents nivells només canvien l'organització lògica de les dades, no les transformen.
- Només es fa una lectura per cada item, pel que el overhead de la indirecció és el mateix que si es copiessin.

4.3.1 Mètode

Fins a aquest punt s'han anat prenent decisions sobre les propietats de l'arbre. Ara es procedeix a definir un mètode exhaustiu per definir quins seran i com han d'estar organitzats els nodes de l'arbre. Aquest arbre ha d'unificar les estructures que es fan servir en tot el procés de manipulació.

1. **Desglossar les estructures intermitges.** Cal doncs, analitzar les diferents etapes i enumerar-les.
2. **Mirar l'estructura individual.** Descriure les dades intermitges. Una forma adequada de fer-ho serà extraient les claus per les que es pot accedir a les dades intermitges.
3. **Fusionar les estructures.** Agrupar les diferents estructures extretes intentant unificar les claus comunes entre elles.

4.3.2 Notes

- Repassar que es poden fer totes les transformacions que calen (e.g. la transformada KLT requereix que tota la imatge estigui en memòria o que es puguin efectuar dues passades [3]).
- Permetre tot o part en memòria. Generar una estructura on hi càpiga tot, però que permeti posar a nul les parts d'aquesta que no s'utilitzaran.
- Si cal desambiguar ordre, posar més amunt a l'arbre la forma d'accedir més comú. i.e. $k1$ i $k2$ són claus i es vol accedir a les dades per $k1 = a$. Es pot organitzar l'arbre de dues maneres:

	Estructura	**Forma d'accedir**
a)	arrel $\xrightarrow{k1}\bullet\xrightarrow{k2}\bullet$	`Foreach(x←arrel(k1=a)) { codi }`
b)	arrel $\xrightarrow{k2}\bullet\xrightarrow{k1}\bullet$	`Foreach(y←arrel) { x=y(k1=a); codi }`

El cas b) és pitjor ja que requereix una operació més per iteració. Cal tenir present que aquest és només un criteri de desambiguació i que el seu impacte a l'execució d'un programa pot ser mínim.

4.3.3 Etapes

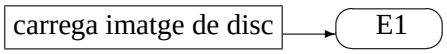
1. Carregar imatge
2. Tiling
3. DC Level
4. Multicomponent transform
5. Transformada wavelet
6. Separació en codeblocs
7. Procés a nivell de bloc

4.3.4 Estructures intermitges

La notació per descriure les claus d'accés a les dades és la següent:

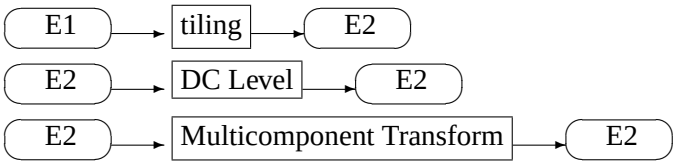
Símbol	Descripció
$\{a_1, \ldots, a_n\}$	Indica que les claus a_i permeten accedir a les dades. No és necessari usar les n claus per identificar una dada de forma única. Tampoc tenen relació d'ordre entre elles.
$(a_1, \ldots, a_n)$	Indica que les claus a_i permeten accedir a les dades. Tenen relació d'ordre entre elles. Generalment és necessari usar les n claus per identificar una dada de forma única.

Estructura 1



$$Keys(E1) = \{x, y, component\}$$

Estructura 2

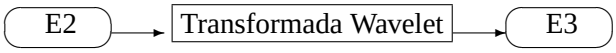


$$Keys(E2) = \{x, y, component, (tile, \{x_{tile}, y_{tile}\})\}$$

Nota: Els dominis de la clau de component són diferents abans i després d'una transformació de components.

$$component \in keys(E2), component \in keys(E2'), Domini(component) \neq Domini(component')$$

Estructura 3



$$Keys(E3) = \{x, y, component, (tile, \{x_{tile}, y_{tile}, (resolutionlevel, subband,$$

$$\{x_{subband}, y_{subband}\})\})\}$$

Estructura 4

$$Keys(E4) = \{x, y, component, (tile, \{x_{tile}, y_{tile}, (resolutionlevel,$$

$$\{(subband, \{x_{subband}, y_{subband}\}), precint\}, codeblock, \{x_{codeblock}, y_{codeblock}\})\})\}$$

4.3.5 Arbre resultant

El resultat de seguir el mètode anteriorment descrit és el següent:

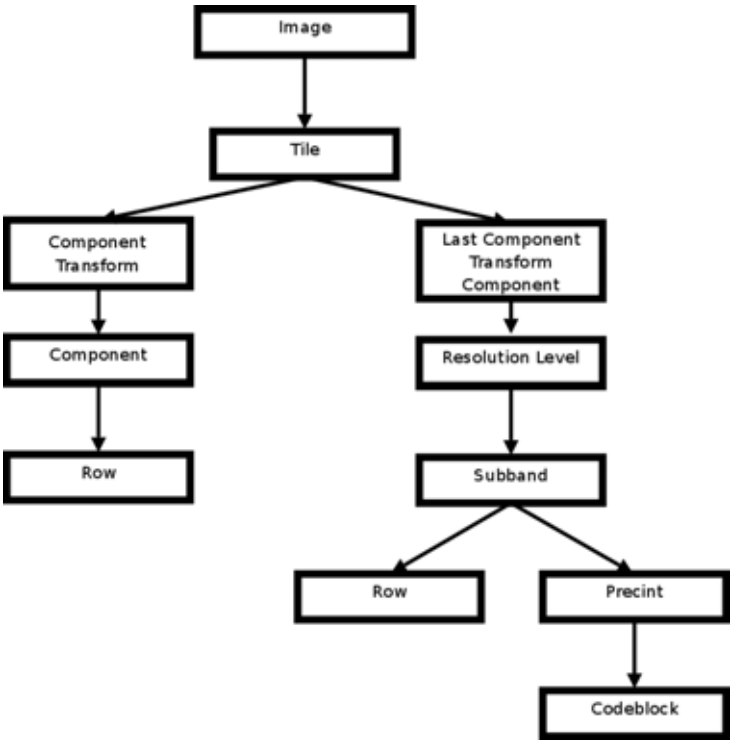


Figura 4.3: Estructura abstracta de les dades

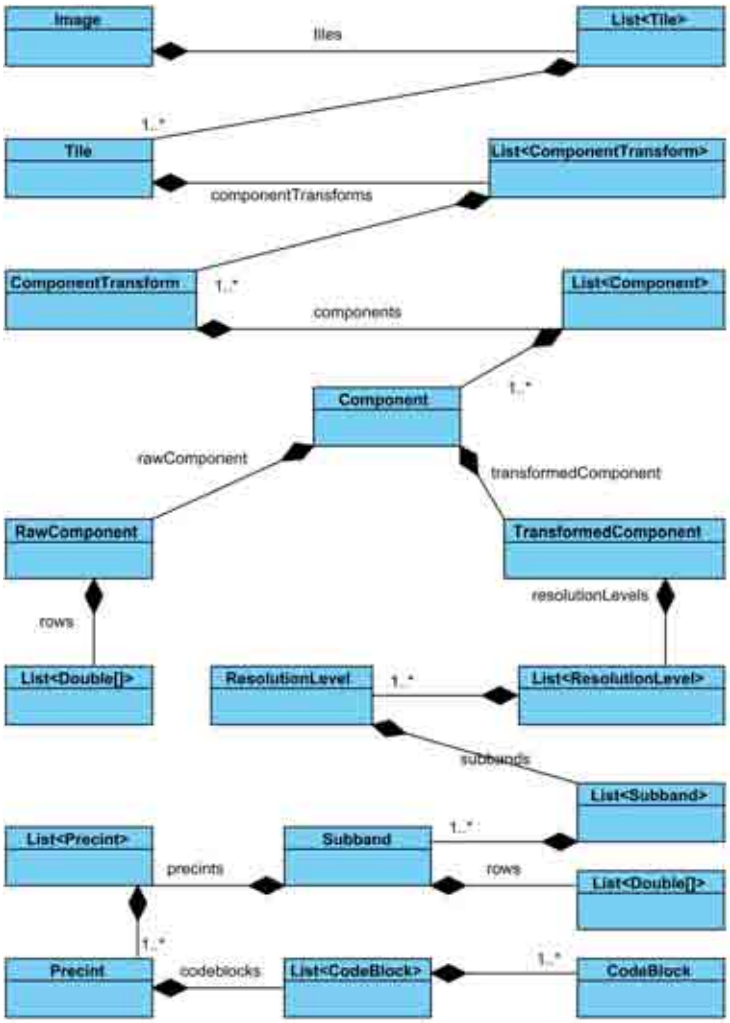


Figura 4.4: Diagrama uml de l'estructura de dades

4.4 Recursos de càlcul

En aquesta secció es fa una revisió dels recursos de càlcul que cal implementar per a realitzar les diverses operacions en les que calen operacions matemàtiques complexes (i.e. les operacions organitzatives estan relacionades intrínsecament amb el director que correspongui). S'ha fet especial èmfasi en la implementació de la transformada wavelet, ja que aquesta limita la possibilitat de treballar amb memòria limitada segons el disseny. En canvi la resta de recursos, poden funcionar com a caixes estanques i es pot refactoritzar el codi existent a BOI 1 per a implementar-los.

4.4.1 Qüestions generals

Tot seguit un parell de qüestions sobre la interacció dels recursos de càlcul amb l'exterior. Aquestes qüestions fan especial referència a com els recursos de càlcul dipositen els resultats, ja que si els recursos no poden deixar els resultats directament sobre l'estructura de dades, els directors corresponents han d'extreure aquests recursos de forma puntual i observar quan finalitzen les operacions.

Problema: On es guarda el resultat i com s'indica? No tenim punter a punter.
Decisió: Com que no hi ha punter a punter els resultats es poden obtenir a) per un punter/-referència a pare + índex de fill o b) per valor de retorn.
Arguments:

- En el cas de a) implicaria que els mòduls haurien de tenir un mínim coneixement de l'estructura de les dades.
- En algun moment el director haurà de reprendre el flux d'execució dels mòduls, així que és factible b).

Problema: Rebre més d'un resultat per crida.
Solució: Fer una crida de processament calcular() i fer diferents crides per obtenir els resultats: obtenirResultat1(), obtenirResultat2(), ...

4.4.2 Transformada de Components

Es tracta d'una unitat estanca pel que fa a les interaccions amb les altres unitats. Aleshores el seu disseny es regeix per les regles plantejades a l'apartat 4.4.1.

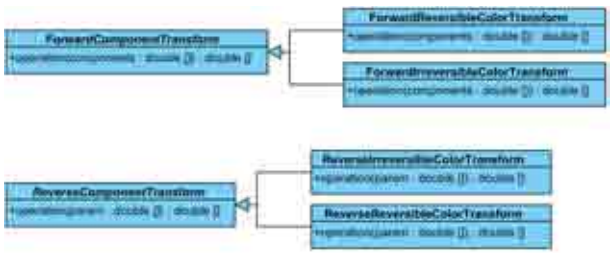


Figura 4.5: Diagrama de classes de la transformada de components

4.4.3 Transformada Wavelet

La implementació de la transformada wavelet és molt important de cara a poder efectuar organitzacions que milloren una implementació de JPEG 2000 (i.e. la transformada local i la transformada amb memòria limitada). A continuació es presenta una implementació que permet realitzar totes aquestes operacions. Aquesta implementació està basada en elements reutilitzables en els que es configuren amb un flux d'entrada (imatge original) i dos de sortida(sub-banda baixa, sub-banda alta). Aquests elements es poden combinar amb una agrupació d'aquests per efectuar la transformada vertical de forma paral·lela. Com que les operacions es realitzen en forma de flux no cal disposar de totes les dades en memòria per realitzar l'operació (memòria limitada). I també degut a la seva flexibilitat d'organització es poden construir tot tipus de transformades.

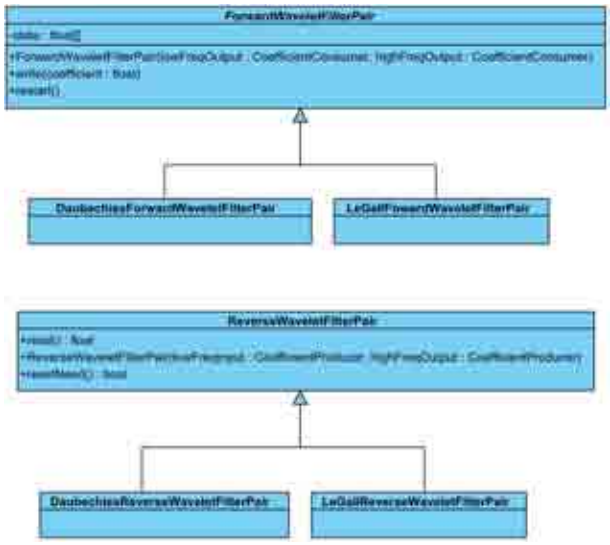


Figura 4.6: Diagrama de classes de les parelles de filtres usats en les transformades wavelet

El model de transformada endarrera és el mateix que la transformada endavant en el concepte d'usar fluxos de dades i unitats reusables ,però evidentment realitza l'operació en el sentit contrari.

Problema: Paral·lelitzar la transformada wavelet amb els block coders.
Arguments:

- La transformada wavelet funciona per push i no per pull, però cal saber quan va produint resultats (cada push no és un resultat). Cal saber-ho perquè a mesura que es van produint s'han de passar a la següent etapa per poder fer la compressió amb memòria limitada.

Solució: Muntar un helper que porti una cua on s'hi guardi una identificació a un bloc resultat de la transformada wavelet. Així a mesura que es van produint blocs, el recurs de la transformada els va afegint a la cua. Cada cop que el director fa un push al recurs de la transformada wavelet, comprova l'estat de la cua. Si hi troba elements els envia a processar.

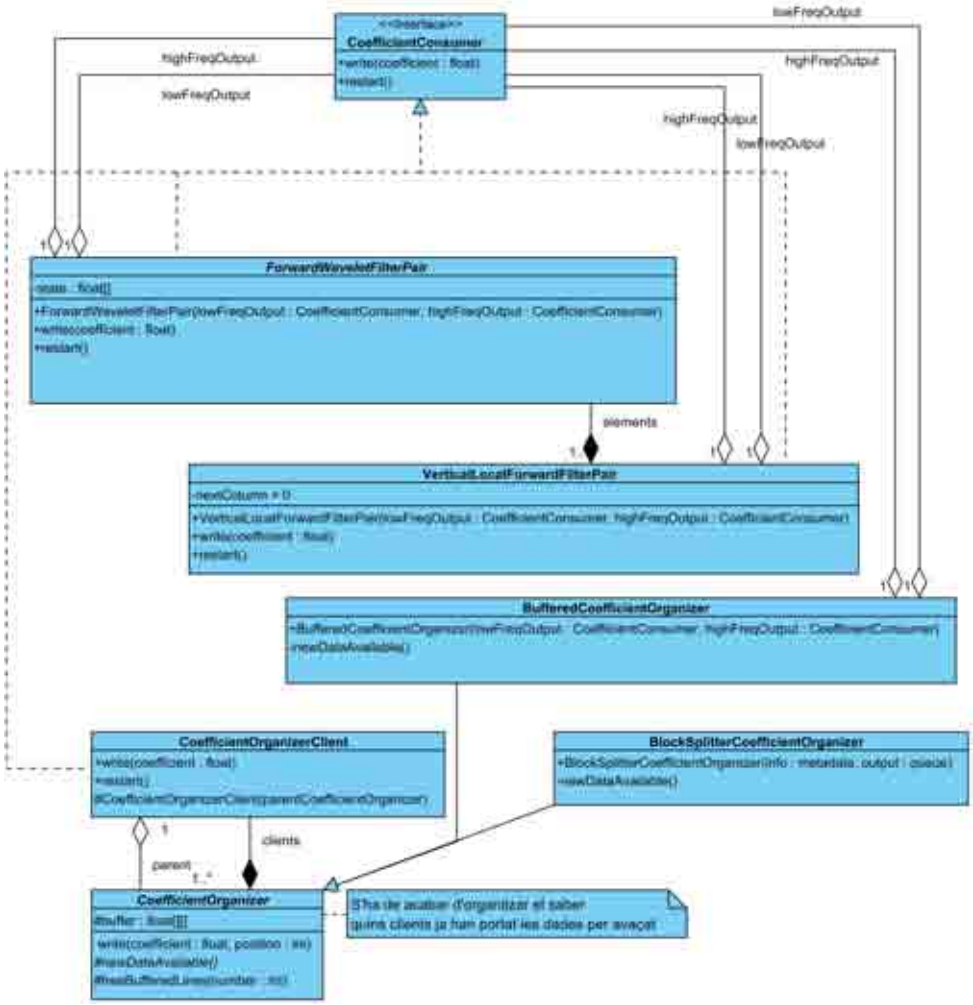


Figura 4.7: Diagrama de classes de la transformada wavelet endavant

4.4.4 Quantització

Es tracta d'una unitat estanca pel que fa a les interaccions amb les altres unitats. Aleshores el seu disseny es regeix per les regles plantejades a l'apartat 4.4.1.

4.4.5 Codificació dels plans de bits

Es tracta d'una unitat estanca pel que fa a les interaccions amb les altres unitats. Aleshores el seu disseny es regeix per les regles plantejades a l'apartat 4.4.1.

4.4.6 Codificació per entropia

Es tracta d'una unitat estanca pel que fa a les interaccions amb les altres unitats. Aleshores el seu disseny es regeix per les regles plantejades a l'apartat 4.4.1.

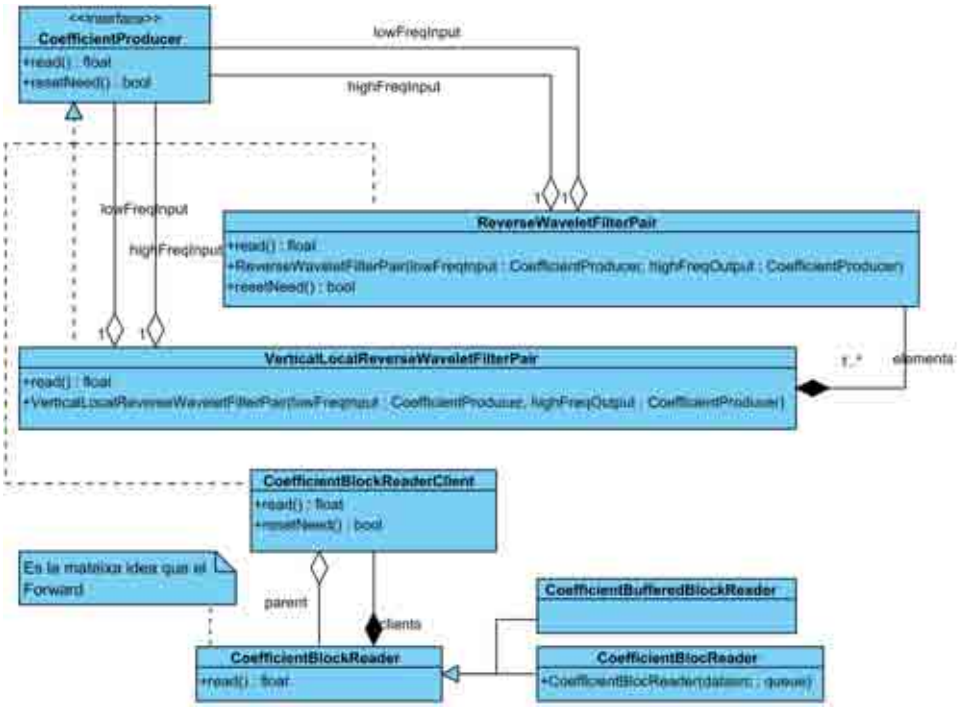


Figura 4.8: Diagrama de classes de la transformada wavelet endarrera

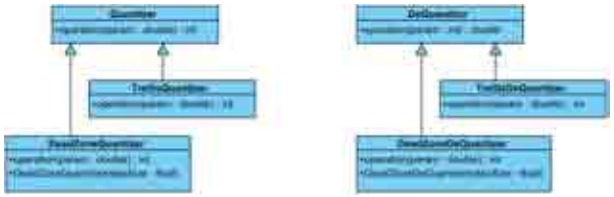


Figura 4.9: Diagrama de classes de l'etapa de quantització

Figura 4.10: Diagrama de classes de les parelles de filtres usats en les transformades wavelet

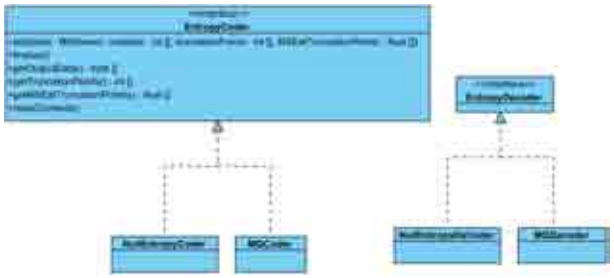


Figura 4.11: Diagrama de classes de les parelles de filtres usats en les transformades wavelet

4.5 Director

El director és el mòdul que organitza el flux d'informació entre les diferents parts de l'estructura de dades, fent les transformacions necessàries en els corresponents recursos de càlcul. Evidentment que no sempre es voldran realitzar les mateixes operacions sobre unes dades. En el cas més evident existeixen dues transformacions que s'hauran de poder fer: comprimir i descomprimir. Però, al ser JPEG 2000 un format ampli que permet força més operacions també es poden pensar en manipulacions de les dades que permetin extreure zones en concret d'una imatge, o que de forma interactiva vagin transmetent la informació. Tot i que aquesta llista de diferents transformacions és força oberta, a continuació es presenten diferents directors on cadascun representa una transformació diferent.

Encara que els directors han de suportar transformacions complexes diferents, hi ha força parts d'aquestes que són compartides i per això es proposa a continuació factoritzar aquestes etapes comunes en els diferents directors.

Compress	Transforma una imatge en format raw a una imatge comprimida en JPEG 2000.
Decompress	Fa el procés invers al director anterior.
Extract	Extreu una zona puntual d'una imatge comprimida en JPEG 2000 atenent-se a diferents criteris (e.g. la qualitat).
Eye	Indexa un arxiu i el va proporcionant als seus clients de forma interactiva.

Taula 4.2: Tipus de directors

BuildTree	Construeix una estructura de dades buida que s'omplirà en etapes posteriors.
PopulateFromFile	Omple una estructura de dades des d'un arxiu.
DumpToFile	Realitza l'operació contrària a PopulateFromFile.
IndexFile	Indexa els continguts d'un arxiu JPEG 2000 a l'estructura de dades global, de forma que quan es demanen unes dades es pot accedir a elles fàcilment.
CompressZone	Donades unes especificacions concretes i unes dades delimitades es comprimeixen aquestes dades.
DecompressZone	El procés invers a l'operació anterior.

Taula 4.3: Tipus d'etapes

4.5.1 Gestor de paràmetres

Respecte de la relació del director amb el gestor de paràmetres s'ha arribat a la següent conclusió:

Decisió: És el Director qui accedeix al gestor.

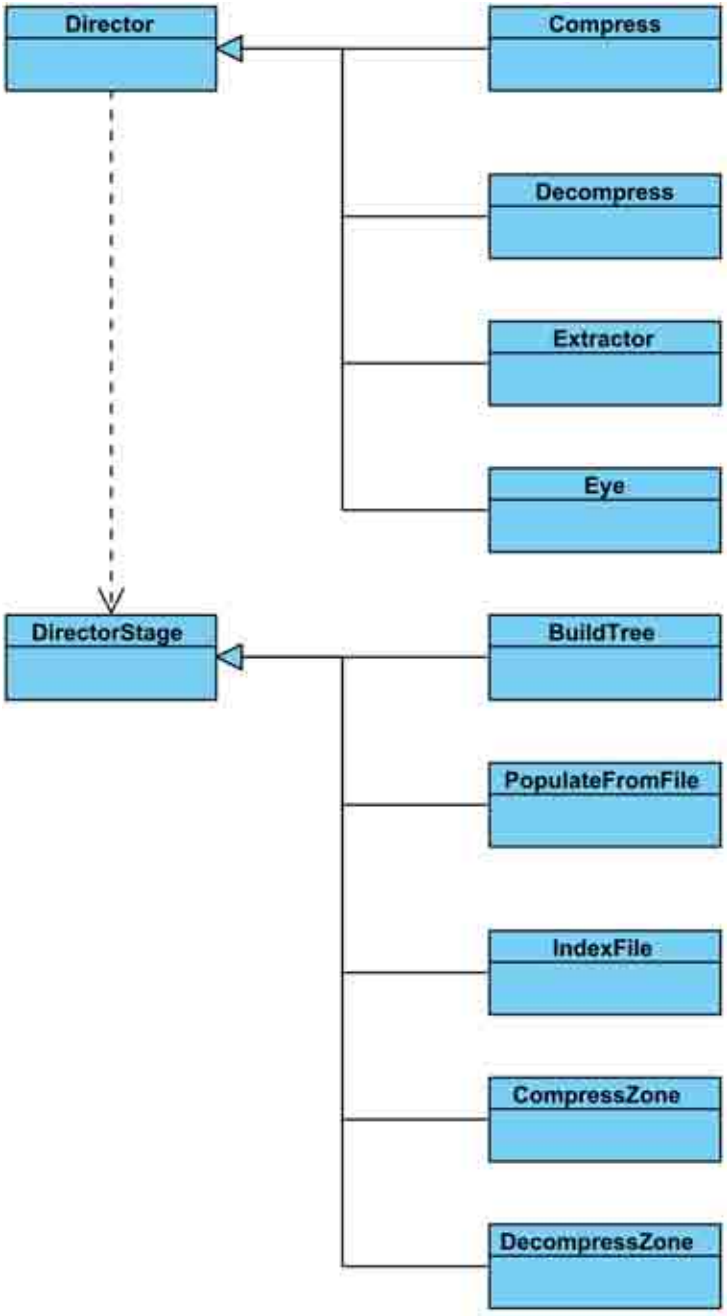


Figura 4.12: Esquema UML dels directors

Arguments: Pot ser que un mòdul hagi de disposar de diferents configuracions. Si hagués de poder decidir quina li cal en cada moment hauria de tenir consciència del flux global d'execució que trenca el principi organitzatiu del director.

4.6 Gestor de paràmetres

4.6.1 Motivació

Un dels requisits del projecte és que s'ha de disposar d'un gestor de paràmetres de configuració.

- Aquest gestor ha de poder parsejar paràmetres des de la línia de comandes.
- L'especificació dels paràmetres que es poden passar, els tipus, els rangs, les dependències o les comprovacions addicionals que requereixin s'han de poder especificar en un mòdul independent de la funcionalitat de parsejar la línia de comandes.

4.6.2 Requisits de disseny

- Ha de ser separable. S'han de poder especificar els paràmetres que ha de parsejar el gestor de forma independent per cada mòdul. Tot i així, s'ha de tenir en compte que hi poden haver mòduls amb dependències de paràmetres.
- Ha de ser fàcil de codificar, ja que hi ha molts paràmetres i això fa que una diferència aquí sigui molt rellevant.

4.6.3 Decisions

Desició: Usar a) Obj.getParameter1() o b) Obj.getParameter("1") per accedir a les variables de configuració.
Arguments:

- Amb a) es fa la verificació de tipus totalment en temps de compilació
- Amb b) es pot fer Obj.getBoolParameter("x") que fa la meitat de les comprovacions i l'altre meitat les fa al accedir en runtime.
- Amb a) s'ha de programar una funció per paràmetre.

El tercer argument fa que sigui molt pesat haver de gestionar els paràmetres, i decanta la tria a b). Tot i això b) encara presenta el següent problema: A b) hi poden haver peticions de paràmetres que no existeixen o que no es llegiran mai. I només es detectaran en temps d'execució. Possibles solucions per aquest problema són:

- Repassar l'ús de tots els paràmetres i emetre warnings pels que no s'hagin usat.
- Usar eines com coverlipse per assegurar que les línies que accedeixen a la configuració han estat cobertes per almenys algun camí. Només és necessari que que s'hagin cobert per un dels camins d'execució per fer la verificació de tipus, o verificar que els paràmetres existeixen.

Decisió: Usar arxius d'especificació de paràmetres amb .java amb syntax "encoberta". Es tracta de fer que la syntaxs dels arxius d'especificació de paràmetres sigui una subsyntax de java.

Arguments:

- No cal fer un parser pels arxius d'especificació del gestor de paràmetres, ja que s'aprofita el del compilador de java.
- Molts menys problemes per la detecció d'error que en la utilització d'un parser propi, ja que els compiladors tenen un sistema de detecció d'errors força bo.
- Permet incrustar codi directament per més extensibilitat a l'hora de, per exemple, fer comprovacions addicionals sobre paràmetres.
- Cada especificació del gestor de paràmetres és un objecte java.
- Caldrà un petit header pels arxius de especificació perquè compilin.

Desició: S'han de poder carregar més d'un arxiu d'especificació de paràmetres, pel mateix gestor de paràmetres.

Arguments:

- Així, cada funcionalitat pot tenir els seus paràmetres de configuració per separat i aleshores es poden reaprofitar els arxius d'especificació per diferents directors en les funcionalitats compartides que tinguin.

4.6.4 Disseny

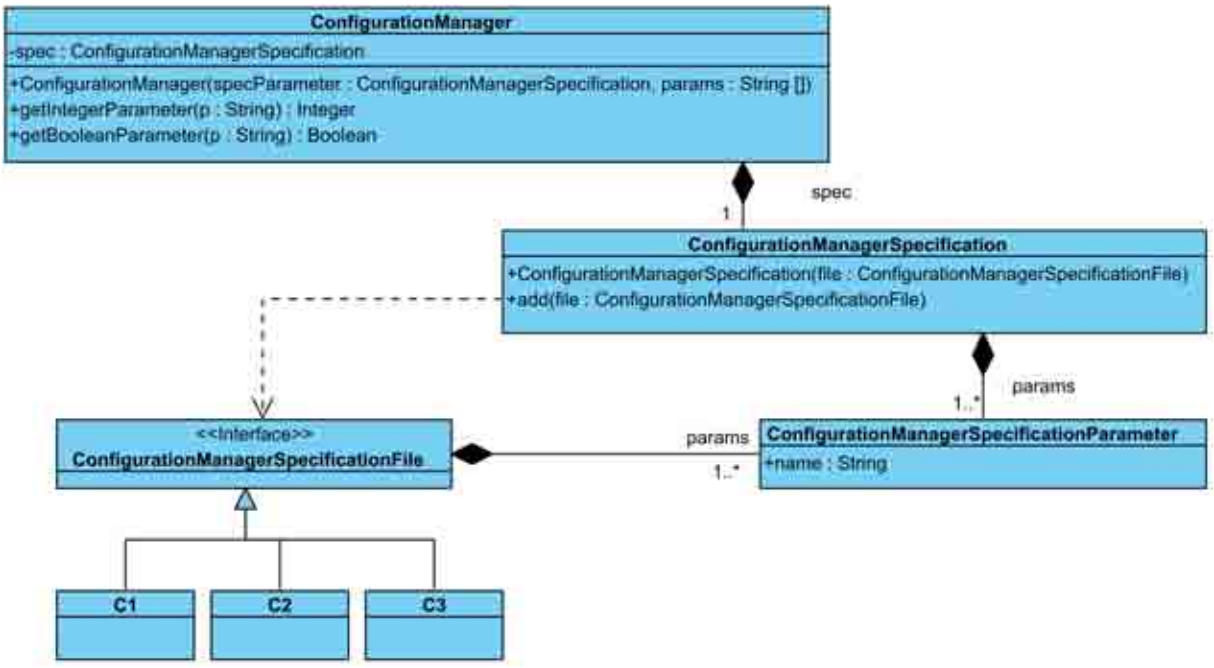


Figura 4.13: Diagrama de classes del gestor de paràmetres

4.7 Resum

En aquest capítol s'han exposat els diferents dissenys que pretenen encarrilar la versió 2.0 de BOI. S'han detallat primer els mòduls en els que es descomposa el disseny, i després s'han anat definint aquests de forma individual. Sobretot s'ha fet especial èmfasi en les intercomunicacions entre les diferents parts, aïllant així la complexitat d'aquestes. En cada etapa s'ha començat raonant sobre les decisions de disseny que s'han pres i enumerant els arguments que s'han trobat en contra i a favor d'aquestes. Després d'una profunda reflexió plasmada en els arguments s'ha procedit a definir els dissenys.

En referència a les qüestions generals dels genèrics i els tipus de dades de doble precisió s'han pres dues decisions. La primera, sobre els genèrics ha estat que tot i que l'àmbit d'aplicació és gran, l'overhead ho és més i no s'usarien. I la segona, sobre els tipus de doble precisió que tenen un impacte nul en les operacions de punt flotant(en punt fix sí que hi ha impacte i és gran) i que la única penalització sorgia del fet que ocupen més memòria i per tant augmenten les faltes de cache i carreguen més el bus de memòria. Tot i que aquesta última penalització és totalment acceptable.

Pel que fa a l'organització de les dades, el raonament ha estat diferent i s'ha seguit un procediment constructiu per arribar a l'estructura que s'ha presentat, per tal de fonamentar-la sòlidament.

En la resta de les etapes les decisions que s'han anat prenent han seguit un caire més específic o inclús, no n'han calgut de decisions específiques per les etapes que s'han pogut encapsular bé.

Capítol 5

Planificació i costos

A l'inici del projecte es va realitzar una planificació temporal de les tasques. Aquesta es presenta a continuació en comparació amb el que s'ha produït realment i una valoració en comparació econòmica.

5.1 Tasques

La planificació es va dividir en tasques simples. Aquestes es presenten tot seguit:

Planificació Es definiran les etapes i s'elaborarà la planificació inicial del projecte.

Documentació i formació El projectista es documentarà i es formarà en els àmbits del seu projecte de final de carrera.

Anàlisi de requisits Es definiran els requisits tant funcionals com no funcionals del disseny a elaborar. Pot incloure requisits de grups coll·laboradors amb GICI.

Mòduls S'establiran els diversos blocs principals del disseny seguint els criteris de alta Cohesió (funcional) i baix Acoblament (de Dades).

Classes S'elaboraran els diferents diagrames que concretaran la implementació fins al nivell de classe.

Detallat (Opcional) S'especificaran les funcions i variables de les classes (sempre que aquestes siguin rellevants al disseny).

Memòria És documentarà el treball fet.

Cal tenir en compte que l'etapa de formació va començar molt abans que el projecte de fi de carrera amb l'assistència del projectista als seminaris del GICI durant un any.

5.2 Terminis

A continuació es presenta la taula amb els terminis que es van estimar a l'apèndix B i els terminis que finalment han estat, juntament amb la variació de la previsió amb el real.

Els terminis en general s'han pogut flexibilitzar, sense que fos necessari sortir-se gaire de la planificació i igualment poder entregar la memòria en el termini previst.

	Previst		Real		
Tasca	Inici	Fi	Inici	Fi	Δ
Planificació	-	-	-	-	-
Documentació i Formació	-	19/02/2007	-	19/02/2007	-
Anàlisi de requisits	19/02/2007	12/03/2007	19/02/2007	19/03/2007	7 dies
Mòduls	12/03/2007	26/03/2007	19/03/2007	26/03/2007	-7 dies
Classes i Detallat	26/03/2007	21/05/2007	26/03/2007	14/05/2007	-7 dies
Memòria	21/05/2007	01/06/2007	14/05/2007	07/06/2007	13 dies

Taula 5.1: Terminis

	Hores			Cost		
Tasca	Previst	Real	Δ hores	Previst	Real	Δ cost
Planificació	3h	3h	0h	90€	90€	0€
Documentació i Formació	-	122h	-	-	3660€	-
Seminaris Gici	-	50h	-	-	1500€	-
Curs Projecte fi de carrera	-	30h	-	-	900€	-
Lectura articles	10h	12h	2h	300€	360€	60€
Familiarització amb el codi anterior	-	10h	-	-	300€	-
Terminacions òptimes codificador MQ	-	20h	-	-	600€	-
Anàlisi de requisits	5h	4h	-1h	150€	120€	-30€
Mòduls	3h	8h	5h	90€	240€	150€
Classes i Detallat	40h	40h	0h	1200€	1200€	0€
Memòria	30h	48h	18h	900€	1440€	540€
Total	-	225h	-	-	12480€	-

Taula 5.2: Terminis

5.3 Costos

A continuació es presenta el cost que s'estima que ha tingut el projecte realitzat. Aquest projecte ha consistit en manipular un volum gran d'informació molt complexa i traduir-la en un disseny de software. Així doncs el que ha requerit una gran quantitat d'esforç és l'apartat de documentació i formació per tal de poder entendre-la bé. Un cop s'ha entès aquesta informació realitzar els següents passos no ha estat complicat. Tot i així també ha requerit un bon munt de feina realitzar la memòria, per tal de sintetitzar tota aquesta informació de forma coherent.

5.4 Resum

S'han presentat les planificacions tant de cost com de temps, i s'han comparat amb els valors reals que s'han produït. Cal destacar que s'han pogut complir en general els terminis marcats, tot i que en ocasions ha implicat càrregues de feina irregulars.

En referència al cost s'ha vist que no es tracta d'un cost excessivament elevat i que la partida més important ha estat la que ha calgut en formació.

Capítol 6

Conclusions

A continuació es presenten les conclusions del projecte, detallades en tres blocs. Primerament es valorarà l'acompliment dels objectius que tenia el projecte. Després es reflexionarà sobre els resultats o coneixements que s'extreuen d'haver realitzat el projecte. I finalment, es desenvoluparan breument les línies futures d'actuació que continuarien la feina ja presentada.

6.1 Valoració

Recapitulant sobre els objectius expressats a la introducció:

Ajust a l'estàndard De forma qualitativa es pot afirmar que el disseny sí segueix l'estàndard. Però aquest és un objectiu que encara no es pot quantificar. S'ha fet el possible perquè el disseny el seguís, però serà la implementació el que definirà la seva adhesió o no a l'estàndard.

Flexibilitat Aquest és un objectiu que sí que s'ha aconseguit. Es pot valorar clarament per exemple en el desacoblament dels càlculs amb l'estructura de les dades o la independència de la configuració dels recursos de càlcul. I en un exemple més concret, la facilitat d'afegir una altra transformada wavelet, que simplement consistiria en afegir una altra especialització de la classe dels filtres de transformada wavelet.

Memòria limitada El principals obstacles d'aquest objectiu eren l'estructura de dades i la transformada wavelet. En el primer cas s'ha plantejat de forma que no s'hagin d'emmagatzemar totes les dades al mateix moment i per tant no calgui memòria proporcional a la mida de les dades. En el segon, amb l'esquema de flux plantejat se soluciona implícitament aquest problema. Per tant, es pot considerar un objectiu acomplert.

6.2 Resultats

Una de les conclusions que s'extreu d'haver realitzat aquest projecte de final de carrera és sobre l'esperit de la compressió que es realitza en JPEG 2000. És curiós veure com aquest funciona per etapes en les que es va preparant de forma progressiva la compressió. Els dos casos més clars sobre aquesta forma de procedir són l'agrupament d'energia i el Post-Compression Rate/Distortion Optimization.

La transformada de components (e.g. Principal Component Anàlisis) i la transformada wavelet són exemples d'etapes on s'agrupa l'energia de la imatge en zones concretes. Aquest agrupament d'energia el que fa és que es pugui separar el que interessa del que no. Per tant, es pot discriminar millor a l'hora de realitzar compressió amb pèrdua.

En el cas del Post-Compression Rate/Distortion Optimization, intervenen quatre etapes: la quantització, la codificació per plans de bits, la codificació aritmètica i la optimització en si. Primer a l'etapa de quantització es tria un pas suficientment petit, així a l'etapa de codificació per plans de bits, codificar més o menys plans equival a triar un pas més o menys gran. Després a l'etapa de codificació per plans de bits el que es fa és codificar tots els plans de bits, per tot seguit comprimir-los aritmèticament. Finalment, en l'etapa d'optimització es pot realment quantificar el pas del quantitzador simplement despreciant les últimes dades produïdes pel codificador aritmètic.

6.3 Línies futures

La línia de continuació d'aquest projecte és evidentment realitzar la implementació del disseny. Aquesta implementació es divideix en dues línies diferenciades: adaptar els recursos de càlcul de BOI 1.0 a les noves interficies i implementar la lògica dels directors.

L'adaptació dels recursos de càlcul és una tasca que en si no ha de requerir molta feina excepte en aquells punts on l'estructura de les dades sigui diferent. Si que serà important fer èmfasi en assegurar que la conversió de les interficies sigui coherent i en evitar afegir restriccions contractuals. És més, s'hauria de reduir al màxim el disseny per contracte a les interficies dels recursos de càlcul, ja que aquests s'espera que canviïn força menys que els directors que els usin i, els errors en els recursos de càlcul per incompliment de contracte poden ser gairebé impossibles de detectar.

Pel que fa a les estructures de control que són els directors, aquestes s'hauran de deixar com a última etapa possiblement després de la fase de test dels altres mòduls. Això és degut a que es recolzaran fortament en aquestes altres etapes. Per tant, seria bo evitar que es produïssin errors que poguessin provocar modificacions de codi en cascada des de les parts inicials fins al director.

Apèndix A

Codi estudi tipus de dades

```c
#include <stdlib.h>
#include <stdio.h>

void init_params(int * a, int * b, long long * c, long long * d) {
        long long x;
        x =  1 + (long long) (10000000000.0 *
                (rand() / (RAND_MAX + 1.0)));
        *a = *b = x;
        x =  1 + (long long) (10000000000.0 *
                (rand() / (RAND_MAX + 1.0)));
        *c = *d = x;
}

int main() {
        int a, b;
        long long c, d;

        init_params(&a, &b, &c, &d);
        printf("%lld %d\n", (c*d), (a*b));
        return 0;
}
```

Figura A.1: Codi c del test de multiplicació

```
        call    init_params
.LVL3:
        .loc 1 26 0
        movl    -20(%ebp), %edi
.LVL4:
        movl    -24(%ebp), %eax
.LVL5:
        movl    -36(%ebp), %esi
        movl    $.LC5, (%esp)
        imull   %edi, %eax
.LVL6:
        movl    -32(%ebp), %edi
        movl    %eax, 12(%esp)
        movl    -32(%ebp), %eax
        imull   %esi, %edi
.LVL7:
        mull    -40(%ebp)
        leal    (%edi,%edx), %esi
        movl    %eax, %ecx
.LVL8:
        movl    -28(%ebp), %edx
        movl    -40(%ebp), %eax
        movl    %ecx, 4(%esp)
        imull   %edx, %eax
        leal    (%esi,%eax), %ebx
        movl    %ebx, 8(%esp)
        call    printf
```

Figura A.2: Codi ensamblador per a la multiplicació del test de la figura A.1

Apèndix B

Planificació inicial

Planificació del projecte

Continguts

1. Pla del projecte

1.1 Formulació del problema

El Group on Interactive Coding of Images(GICI) de la UAB des de ja fa uns anys porta desenvolupant el software BOI. El BOI és una implementació de l'estàndard JPEG 2000. La implementació actual té alguns aspectes de disseny que es podrien millorar. Aquests canvis al disseny sorgeixen del fet que ara el grup té una visió molt més complerta i en profunditat de l'estàndard.

1.2 Declaració de l'objectiu fi del projecte

El projecte de fi de carrera que es descriu en aquest document té per objectiu elaborar un disseny formal d'implementació de l'estàndard JPEG 2000 tenint en compte els nous coneixements adquirits pel grup.

1.3 Abast del projecte

El projecte cobreix totes les fases del procés de l'enginyeria del software des de l'anàlisi de requisits fins als diagrames de classes. Serà en funció del temps disponible que s'aprofundirà en el nivell de detall dels diagrames de classes.

1.4 Plantejament a seguir

El projecte tindrà una part prèvia, on el projectista es documentarà. Després es procedirà amb el procés de l'enginyeria del software pròpiament. S'establiran punts de control a les diferents etapes del procés per validar-ne els resultats (inclosa l'etapa de documentació).

1.5 Especificacions del producte final

Les especificacions del contingut dels diagrames es pactaran a l'etapa d'anàlisi de requeriments. L'aspecte formal dels diagrames seguirà el model UML tant com sigui possible.

1.6 Estructuració del treball

El treball disposarà de 5 etapes i una opcional en funció del calendari. Les etapes són les següents:

- Documentació: el projectista es documentarà.
- Requisits: es definiran els requeriments tant funcionals com no funcionals del disseny a elaborar. Pot incloure requisits de grups col·laboradors amb GICI.
- Mòduls: s'establiran els diversos blocs principals del disseny seguint els criteris d'alta Cohesió (funcional) i baix Acoblament (de Dades).
- Classes: s'elaboraran els diferents diagrames que concretaran la implementació fins al nivell de classe.
- Detallat (Opcional): s'especificaran les funcions i variables de les classes (sempre que aquestes siguin rellevants al disseny).
- Memòria: es documentarà el treball fet.

1.7 Recursos necessaris

Els dos principals recursos seran les eines de modelatge i la documentació sobre l'estàndard JPEG 2000. A l'hora de triar eina de modelatge es valoraran el fets:

- L'efectivitat de l'eina.
- Que sigui software lliure(opció preferida) o que la universitat en tingui una llicència.
- En el cas que sigui una aplicació propietària, que els diagrames generats siguin compatibles amb eines de software lliure.

1.8 Sistema de control

S'establiran reunions de control les setmanes següents per fer el seguiment dels punts de l'estructura:

- 19/2 - Documentació
- 12/3 - Requeriments
- 26/3 - Mòduls
- 21/5 - Classes i Detallat
- 1/6 - Memòria

1.9 Elements de risc

El principal element de risc és el temps previst perquè el volum de feina no sigui l'ajustat.

2. Estudi de viabilitat

2. Objectius de l'estudi

L'objectiu de l'estudi és determinar la viabilitat del projecte de fi de carrera anteriorment exposat.

2.1 Introducció

Dins del marc de grup de recerca GICI (Group on Interactive Coding of Images) de la UAB, s'està treballant en l'estàndard JPEG 2000. El grup ja té una implementació de l'estàndard anomenada BOI. La implementació es va fer per entendre el funcionament del JPEG 2000. Ara, s'ha detectat que hi ha decisions de disseny, que es van prendre quan encara no es tenien tots els coneixements de ja haver fet una implementació, que es podrien canviar.

2.2 Objectius del Projecte

El projecte de fi de carrera que es descriu en aquest document té per objectiu elaborar un disseny formal d'implementació de l'estàndard JPEG 2000 tenint en compte els nous coneixements adquirits pel grup.

2.3 Descripció de la situació inicial

A l'inici del projecte existeix un munt d'experiència sobre la implementació de BOI. És molt important tenir-ho en compte tant a l'hora de definir els nous requisits com a l'hora de buscar inspiració. També poden ser origen d'inspiració altres implementacions, tot i que potser no tindran els mateixos objectius.

Tot i així, el projecte es basa en tornar a dissenyar l'arquitectura, qüestionant tots els passos que es prenen i per tant, no es poden aprofitar els dissenys actuals.

2.4 Especificacions del sistema i de les aplicacions

L'entregable que es desenvolupa en aquest projecte és un disseny d'una implementació de JPEG 2000. Els requisits d'aquest disseny són que s'efectuï de forma coherent amb l'enginyeria del software i que realment serveixi com a punt de partida per una implementació posterior.

2.5 Viabilitat tècnica

El projecte té la gran complexitat tècnica d'implementar un estat de l'art de compressió d'imatges. Aquesta complexitat fa que sigui molt difícil preveure el temps que es dedicarà a cada etapa i per tant és un dels principals factors de risc. Per gestionar aquest risc s'establiran controls a cadascuna de les etapes, tant per complir amb el calendari com per assentar els resultats fins al moment.

2.6 Viabilitat operativa

Aquesta fase no s'efectua en aquest projecte.

2.7 Viabilitat econòmica

És un projecte sense retribució econòmica, així que segur que econòmicament és viable.

2.8 Viabilitat legal

Es tracta d'un disseny d'un estàndard royalty free, per tant és difícil que apareixin problemes legals. A més al limitar el projecte al disseny i no a la implementació, no poden aparèixer problemes de patents.

2.9 Alternatives del projecte

S'ha plantejat un punt opcional a l'estructuració del treball. Aquest punt ha de permetre gestionar un possible excés de feina.

Bibliografia

[1] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1(2):205–220, April 1992.

[2] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *The Journal of Fourier analysis and applications*, 4(3):245–267, 1998.

[3] Q. Du and J. E. Fowler. Hyperspectral image compression using JPEG2000 and principal component analysis. *IEEE Geoscience and Remote Sensing Letters*, 4:201–205, April 2007.

[4] J. E. Fowler and J. T. Rucker. *Hyperspectral Data Exploitation: Theory and Applications*, chapter 3D Wavelet-Based Compression of Hyperspectral Imagery, pages 379–407. John Wiley and Sons, Inc, Hoboken, NJ, 2007.

[5] Intel. *Intel 64 and IA-32 Architectures Optimization Reference Manual*, chapter Appendix C: Instruction Latency and Throughput, page 452. Intel, March 2007. http://www.intel.com/design/processor/manuals/248966.pdf.

[6] JPEG 2000 image coding system - Part 1: Core coding system, Information technology 15444-1. International Organization for Sandardization / International Electrotechnical Commission (ISO/IEC); International Telecommunication Union-Telecom Standardization (ITU-T), December 2000.

[7] JPEG 2000 image coding system - Part 2: Extensions, Information technology 15444-2. International Organization for Sandardization / International Electrotechnical Commission (ISO/IEC); International Telecommunication Union-Telecom Standardization (ITU-T), May 2004.

[8] JPEG 2000 image coding system - Part 3: Motion JPEG 2000. Information technology 15444-10. International Organization for Sandardization / International Electrotechnical Commission (ISO/IEC); International Telecommunication Union-Telecom Standardization (ITU-T), 2005.

[9] JPEG 2000 image coding system - Part 9: Interactivity tools, APIs, and protocols. Information technology 15444-9. International Organization for Sandardization / International Electrotechnical Commission (ISO/IEC); International Telecommunication Union-Telecom Standardization (ITU-T), December 2005.

[10] JPEG 2000 image coding system - Part 10: Volumetric JPEG2000. Information technology 15444-10. International Organization for Sandardization / International Electrotechnical Commis-

sion (ISO/IEC); International Telecommunication Union-Telecom Standardization (ITU-T), March 2007. Final Draft, International Standard.

[11] P. Kulkarni, A. Bilgin, M.W. Marcellin, J.C. Dagher, T. Flohr, and J. Rountree. Reduced memory multi-layer multi-component rate allocation for JPEG2000. In Amir Said and John G. Apostolopoulos, editors, *Image and Video Communications and Processing 2005*, volume 5685, pages 139–150. SPIE, SPIE & IS&T, March 2005.

[12] M.W. Marcellin, M.A. Lepley, A. Bilgin, T.J. Flohr, T.T. Chinen, and J.H. Kasner. An overview of quantization in JPEG2000. *SP:IC*, 17(1):73–84, January 2002.

[13] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, December 1993.

[14] A. Skodras, C. Christopoulos, and T. Ebrahimi. The JPEG 2000 still image compression standard. *IEEE Signal Processing Magazine*, 18(5):36–58, September 2001.

[15] W. Sweldens. The lifting scheme: a new philosophy in biorthogonal wavelet constructions. In *Wavelet Applications in Signal and Image Processing III*, volume 2569, pages 68–79, September 1995.

[16] D.S. Taubman. Kakadu software. http://www.kakadusoftware.com/, 2000.

[17] D.S. Taubman and M.W. Marcellin. JPEG2000: Standard for interactive imaging. *Proceedings of the IEEE*, 90(8):1336–1357, August 2002.

Firmat: Ian Blanes

Bellaterra, Juny de 2007

Resum

JPEG 2000 és un estàndard de compressió d'imatges que utilitza tècniques estat de l'art basades en la transformada wavelet. Els principals avantatges són la millor compressió, la possibilitat d'operar amb dades comprimides i que es pot comprimir amb i sense pèrdua amb el mateix mètode. BOI és la implementació de JPEG 2000 del Grup de Compressió Interactiva d'Imatges del departament d'Enginyeria de la Informació i les Comunicacions, pensada per entendre, criticar i millorar les tecnologies de JPEG 2000. La nova versió intenta arribar a tots els extrems de l'estàndard on la versió anterior no va arribar.

Resumen

JPEG 2000 es un estándar de compresión de imágenes que utiliza técnicas estado del arte basadas en la transformada wavelet. Sus principales ventajas son la mejor compresión, la posibilidad de operar con datos comprimidos y que permite comprimir con y sin pérdida con el mismo método. BOI es la implementación de JPEG 2000 del Grupo de Compresión Interactiva de Imágenes del departamento de Ingeniería de la Información y las Comunicaciones, pensada para entender, criticar y mejorar las tecnologías de JPEG 2000. La nueva versión intenta llegar a todos los extremos del estándar donde la versión anterior no llegó.

Abstract

JPEG 2000 is a state-of-the-art image compression standard based on the wavelet transform. Its main advantages are better compression, the possibility of compressed data manipulation and that lossless and lossy compression can be achieved with the same method. BOI is the JPEG 2000 implementation done by the Group on Interactive Coding of Images of the Information and Communications Engineering department. Its goal is to understand, criticise and improve JPEG 2000 technologies. This new version attempts to go further into the standard.