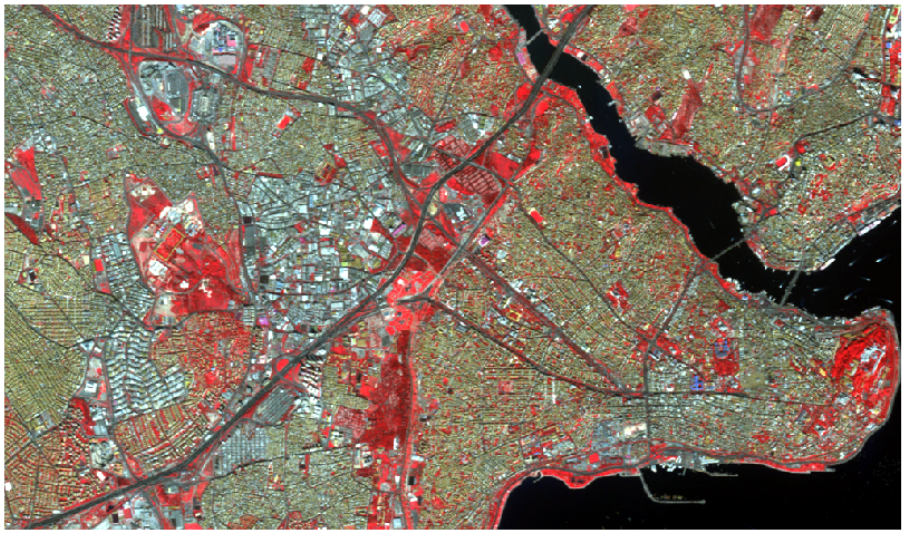
Université de Liège
Faculté des sciences
Département de géographie

Mémoire de fin d'études

Prototypage d'un serveur de données géographiques maillées : Rasdaman



Auteur:
Jean-Michel Begon

Promoteur:
Dr. Jean-Paul Donnay

Année académique 2010-2011

Remerciements

Je tiens à remercier toutes les personnes qui de près ou de loin ont contribué à la réalisation de ce travail.

En particulier :

- ⋆ Mr. Donnay, mon promoteur qui m'a éclairé, guidé, conseillé et soutenu tout au long de l'élaboration de ce mémoire.
- ⋆ Le groupe de discussion de Rasdaman et plus spécifiquement Mr. Baumann pour les nombreuses réponses qu'il a pu me procurer.
- ⋆ Mr. Binard pour les données qu'il m'a fournit.
- ⋆ Mme. Blaise pour les corrections de syntaxe et d'orthographe.
- ⋆ Mr. Laenen pour ses conseils sur la mise en page.
- ⋆ Mes proches pour leur patience, soutien et lectures de ce travail.

Table des matières

Chapitre 1

Introduction

Tous les jours, l'information géographique est plus présente dans nos vies. Le GPS, Google Map (http ://maps.google.com) et autres sont devenus des éléments de notre quotidien. Même la Commission Européenne a décidé, en 2007, de se plonger dans l'information géographique avec la directive *INSPIRE* (http ://inspire.jrc.ec.europa.eu).

Cette montée en puissance de l'information géographique n'est pas sans lien avec la notion de système d'information géographique ou SIG. Le terme de SIG n'est pas récent (Coppock et Rhind 1991) mais n'a pas toujours eu le sens qu'on lui donne actuellement. Un SIG est un système d'information logiciel qui permet d'enregistrer, stocker, modéliser, récupérer, partager, manipuler, analyser et présenter des informations spatiales (Worboys et Duckham 2004).

Un grand nombre d'acteurs ont permis le développement des SIG au fil des ans. Parmi ceux-ci, l'OGC (l'*Open Geospatial Consortium* : www.opengeospatial.org), un organisme international qui vise à la mutualisation de l'information géographique. Les standards publiés par cet organisme sont généralement repris par les concepteurs de SIG.

Néanmoins, les efforts des concepteurs de système d'information géographique se sont souvent orientés vers les informations vectorielles et non vers les informations maillées (ou *raster* en anglais). Pourtant les informations maillées, en terme de masse, sont bien plus importantes que les autres informations (Furtado *et al.* 1997, Reiner et Hahn 2004). Les données maillées ne se rencontrent pas que dans le domaine de la géomatique, mais dans son cadre, sont notamment constituées des images satellites et aériennes, des cartes numériques, des résultats de simulations géoréférencées et des champs géographiques. Les champs géographiques sont des phénomènes qui varient continûment dans l'espace (Donnay 2005), comme la température ou l'altitude.

L'information géographique maillée a souvent été l'intérêt du traitement d'images numériques et de l'algèbre de cartes (Tomlin 1991) alors que les SIG s'en sont peu souciés. Le résultat est que beaucoup de logiciels sont capables de traiter l'information maillée mais que celle-ci se présente sous forme de fichiers sur un disque dur (Libkin, Machlin et Wong 1996, Reiner *et al.* 2002, Gertz *et al.* 2006). L'organisation des informations est alors plus délicate qu'au sein d'un système de gestion de base de données (SGBD) et il n'existe pas de langages de

requêtes permettant de retrouver rapidement l'information pertinente. Dans une base de données spatiales gérant les informations maillées, la requête suivante devrait être possible sans le moindre effort :

> « Retrouve les images satellites qui ont été prises le 12 mai 2011 au dessus de la ville de Liège et dont l'extension spatiale couvre 10 km^2. »

Un SIG pourrait aller encore plus loin dans la complexité des requêtes. On comprend dès lors l'intérêt à passer des systèmes de fichiers aux SIG. Grâce à sa meilleure gestion de l'information, un SIG offrirait aussi une communication plus aisée de l'information. Comme par exemple, l'accès à distance en se connectant au serveur de données ou l'utilisation de *viewer* en ligne.

La bonne gestion des informations géographiques maillées est donc un enjeu primordial. Cette tâche est le rôle d'un composant du SIG appelé le **serveur de données** et est également l'objet de ce mémoire.

Dans ce mémoire nous nous intéressons à un de ces serveurs : Rasdaman (*RASter DAta MANager*, www.rasdaman.org). Notre but sera d'en examiner les capacités .

Ce mémoire s'articule en sept chapitres. Après cette introduction, le second chapitre fait le point sur la normalisation des informations maillées et les systèmes de gestion de l'information s'y intéressant.

Dans le troisième chapitre, on pose l'objectif du mémoire. L'objectif est ensuite reformulé sous forme d'hypothèse. On y développe alors la méthodologie qui permettra de statuer sur l'hypothèse.

Les deux chapitres suivantes sont consacrés au développement des informations nécessaires pour confronter l'hypothèse. Ainsi, le quatrième chapitre fait le tour de la documentation logicielle et scientifique de Rasdaman et dans le cinquième chapitre, on réalise un prototype de serveur de données.

La confrontation avec l'hypothèse fait l'objet du sixième chapitre. On y reprend ce qu'on a appris des sections précédentes et on statue sur l'hypothèse selon la méthodologie établie au troisième chapitre.

Finalement, le septième chapitre conclut ce mémoire en récapitulant l'ensemble de la démarche utilisée et les découvertes réalisées et en discutant des perspectives futures de Rasdaman.

Chapitre 2

État de l'art

Après un bref rappel sur les modes de gestion de l'information géographique, nous allons nous intéresser à deux sujets : la normalisation des informations géographiques maillées et les serveurs de données pouvant gérer cette information.

2.1 Les modes de gestion de l'information géographique

Le but des sciences de l'information géographique est de pouvoir gérer les phénomènes spatiaux. On entend par **phénomène spatial**, tout phénomène qui varie en fonction de la position. Il a donc d'une part des caractéristiques spatiales ou géométriques et d'autre part des caractéristiques attributaires. Le phénomène « commune » a des limites communales mais aussi un certain nombre d'habitants. Le phénomène « température », quant à lui, possède comme attribut la température qui varie en chaque point de l'espace. L'objectif de la science de l'information géographique est donc de gérer cette complexité.

Pour ce faire, cette science a coutume d'organiser l'information en deux catégories en fonction des caractéristiques spatiales du phénomène étudié. La première reprend les phénomènes **spatialement discrets** ; c'est-à-dire qui présentent des caractéristiques identiques sur une portion d'espace. Une commune ou une classe d'altitude sont des exemples de phénomènes spatialements discrets. La seconde catégorie reprend les phénomènes **spatialements continus**. Ceux-ci présentent une valeur différente en chaque point de l'espace. On compte dans cette catégorie la température, la probabilité de trouver un certain minerai, l'altitude, etc.

Les sciences de l'information géographique ont aussi l'habitude de gérer l'information selon deux modes : le mode **vectoriel** et le mode **maillé**. Le premier se base sur des primitives spatiales qui sont le point, la polyligne (un ensemble de segments de droites), le polygone et le solide (pour représenter les phénomènes en trois dimensions). Sur ces primitives, il est possible de construire des structures plus complexes comme des ensembles de points, par exemple. Ces primitives et structures spatiales permettent de capturer l'aspect géométrique du phénomène et d'autres variables gèrent l'aspect attributaire. Le mode vectoriel

s'occupe principalement des phénomènes spatialements discrets. Néanmoins, des amas de points peuvent servir à représenter un sondage spatial de phénomènes spatialement continus.

Le mode maillé, quant à lui, se base sur un ensemble de points ou de cellules répartis de manière régulière. Il s'agira de points organisés comme un sondage systématique s'il est question d'un échantillonage d'un phénomène spatialement continu. Sinon, il s'agira de cellules contiguës de mêmes formes et superficies. C'est le cas des images géographiques pour lesquelles les cellules sont appelées pixels (*Picture Element*).

Le mode vectoriel est bien connu et a déjà fait l'objet d'une standardisation de l'OGC qui a été reprise par l'ISO (ISO 2003). Dernièrement, l'OGC a travaillé sur la notion de *coverage* (OGC 2006a) qui transcende celle d'information maillée. Nous allons maintenant prendre le temps de faire le point sur cette notion et voir où l'information maillée y tient sa place. Nous discuterons ensuite de la normalisation des services web par l'OGC en matière de *coverages* avant de passer aux serveurs de données les gérant ou du moins gérant l'information maillée.

2.2 Normalisation des données géographiques maillées et des services web

Un *coverage* est une fonction qui associe à chaque point de son domaine de définition (spatio-temporel) la valeur du phénomène.

$$coverage \ : \ D \longrightarrow R \ : \ (x, y, t) \longrightarrow a$$

⋆ D est le domaine de définition.
⋆ R est le *range*.

Le domaine spatial du *coverage* décrit l'ensemble des points pour lesquels la fonction est définie. Il est constitué d'un ensemble d'objets spatiaux associés à un système de référence spatiale et/ou temporelle. En général, le domaine est formé par l'ensemble convexe contenant tous les objets. Il s'agit alors d'un *coverage* continu. Il est cependant possible de définir des *coverages* discrets ou discontinus. Le domaine de ceux-ci se limite à l'ensemble des objets spatiaux. La nature des objets géométriques détermine la catégorie de *coverage* (*cf.* 2.2.1).

Le *range* est l'ensemble des valeurs que peut prendre le phénomène. Il peut renseigner une ou plusieurs variables ; plusieurs fonctions peuvent faire l'objet d'un même domaine spatial. Par exemple, un ensemble de variables météorologiques peuvent être renseignées pour un même point.

Le *coverage* continu se base sur une fonction d'interpolation pour déterminer les valeurs du phénomènes en dehors des objets spatiaux constituants le domaine.

Enfin, la classe *coverage* trouve sa place dans la hiérarchie de l'OGC comme sous-classe de la classe générique *feature*. Une *feature* est une abstraction d'un phénomène réel (ISO 2002).

2.2.1 Les sous-classes de *coverage*

Depuis la notion générique de *coverage*, l'OGC a développé la hiérarchie suivante (fig. 2.1) :

- ⋆ *Discrete coverage*
  - *Discrete point coverage*
  - *Discrete grid point coverage*
  - *Discrete curve coverage*
  - *Discrete surface coverage*
  - *Discrete solid coverage*
- ⋆ *Continuous coverage*
  - *Continuous quadrilateral grid coverage*
  - *Thiessen polygon coverage*
  - *Hexagonal grid coverage*
  - *TIN coverage*
  - *Segmented curve coverage*

Nous allons maintenant passer ces classes en revue.

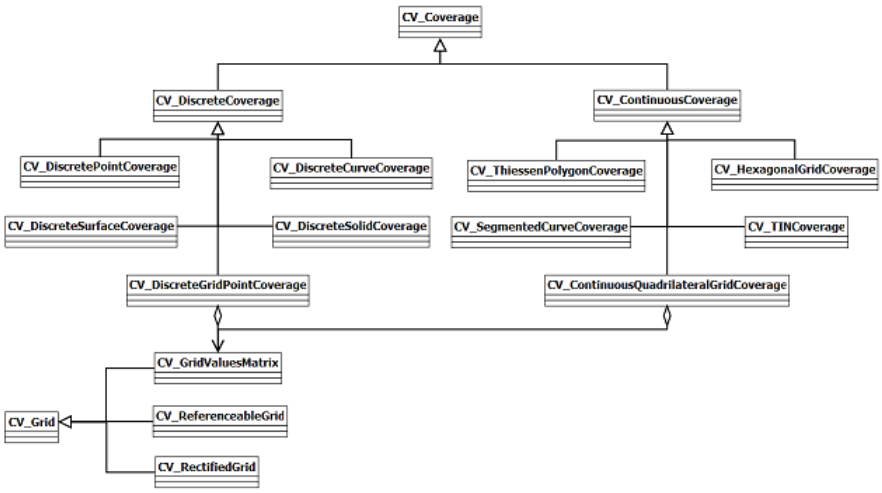


FIGURE 2.1 – Schéma UML de la hiérarchie des classes de *coverages* de l'OGC

2.2.1.1 Les *discrete coverage*

Dans le cas du *discrete coverage*, le domaine de définition n'est pas un ensemble convexe mais une collection d'objets qui ne doivent pas suivre de relations spatiales particulières. La valeur du phénomène n'est disponible que pour ces objets ; il n'y a donc pas d'interpolation. Cette catégorie ne permet pas de représenter des phénomènes spatialement continus.

Hormis le *discrete grid point coverage* (*cfr.* 2.2.1.2) qui est un peu plus particulier, les *discrete coverage* sont des listes d'objets dont seule la dimensionalité change.

2.2.1.2 Le *grid*

Le *grid* est une classe particulière de l'OGC (fig. 2.1) qui ne constitue pas en elle-même un *coverage* mais qui va servir pour l'établissement des *grid coverages*. Elle se base sur une grille constituée d'au moins deux ensembles de lignes.

Ces lignes sont appelées *grid lines*. Les points aux intersections des lignes sont appelés *grid points*. Et les surfaces ou volumes interlignes sont appelés *cells* ou cellules. En particulier, les lignes peuvent devenir des lignes droites et former des parallélogrammes ou des parallélépipèdes. On parlera alors de *quadrilateral grid*. Les écartements entre droites peuvent être uniformes et les droites d'un même ensemble, parallèles, formant une grille régulière. Enfin, les droites s'intersectant peuvent être perpendiculaires. On pourra alors parler de structure *raster*.

Un *quadrilateral grid* peut posséder deux systèmes de coordonnées. Le premier (fig. 2.2) est implicite et toujours présent : il s'agit d'une numérotation des *grid lines*. Les coordonnées permettent donc de faire référence à un *grid point*. Le second est un référentiel externe pour lequel il faut disposer d'informations supplémentaires : la position de l'origine de la grille, l'orientation des axes et l'espacement entre ceux-ci. On parle alors de *rectified grid*. Si le référentiel externe est un référentiel géographique, on parlera de *georectified grid* ou *referenceable grid*. Ces deux types de *grid* (*rectified grid*, *referenceable grid*) constituent les deux premières sous-classes de la classe *grid*.

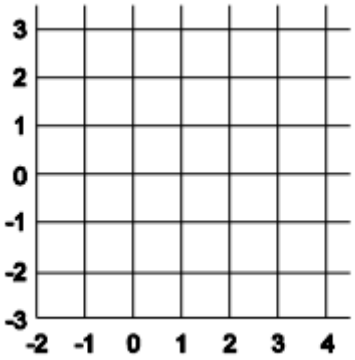


FIGURE 2.2 – Système de référence implicite d'un *quadrilateral grid coverage* (OGC 2006a)

La troisième et dernière sous-classe de *grid* est donnée par la classe *grid value matrix*. Cette classe n'est pas organisée autour d'un système de coordonnées externes mais tire parti de la répartition régulière des points pour les indexer et les stocker de manière séquentielle. Parmi les indexations spatiales prévues par l'OGC se trouvent toutes les indexations classiques (Adam et Gangopadlhyay 1997) : linéaire, en spirale, la diagonale de Cantor, la courbe d'Hilbert-Peano et la courbe de Morton. Cette classe est reprise dans la gestion des *discrete grid point coverage* et des *continuous quadrilateral grid coverage*.

2.2.1.3 Le *continuous quadrilateral grid coverage*

Le *continuous quadrilateral grid coverage* est le penchant en *continuous coverage* du *discrete grid point coverage*. Il est basé sur un *quadrilateral grid*. Le domaine de définition est l'ensemble convexe formé par les *grid points*. Les *grid points* sont stockés via un objet *grid value matrix*. L'interpolation du *range* est de type bilinéaire, biquadratique ou bicubique.

2.2.1.4 Le *Thiessen polygon coverage*

Comme son nom l'indique, le *Thiessen polygon coverage* est un *coverage* reposant sur une tesselation de l'espace basée sur les polygones de Thiessen (Isaaks et Strivastava 1989). En général, le domaine de définition se veut plus grand que l'ensemble convexe des points formant la tesselation ; il faut donc fournir la frontière du domaine. Deux interpolations sont prévues pour ce *coverage* : l'aire perdue et le plus proche voisin. La méthode du plus proche voisin constitue une discrétisassion de l'espace puisqu'elle attribue à chaque point du polygone de Thiessen la valeur du point qui l'a engendré. La méthode de l'aire perdue consiste à redévelopper les polygones de Thiessen en rajoutant le point dont la valeur est recherchée. La valeur au point recherché est une moyenne pondérée des valeurs des points qui ont perdu des surfaces au profit du nouveau point.

2.2.1.5 L'*hexagonal grid coverage*

L'*hexagonal coverage* est un cas hybride du *Thiessen polygon coverage* et du *continuous quadrilateral grid coverage*. Il s'agit d'un *coverage* qui ne traite que du 2D. Il peut être décrit comme une tesselation basée sur un *quadrilateral grid* dont les axes d'un des ensembles sont inclinés de 60° par rapport aux axes de l'autre ensemble (fig. 2.3). Les méthodes d'interpolation sont les mêmes que celles du *Thiessen polygon coverage*. Par contre, l'ensemble de points est géré comme un *continuous quadrilateral grid coverage*.

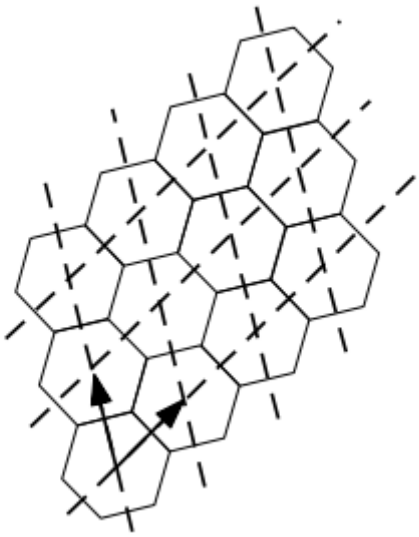


FIGURE 2.3 – Un *hexagonal grid* (OGC 2006a)

2.2.1.6 Le *TIN coverage*

Le *TIN coverage* repose sur un objet de type TIN (*Triangulated Irregular Network* ; GM_TIN ; ISO 2003). La méthode d'interpolation est la méthode classique pour les TIN : barycentrique.

2.2.1.7 Le *segmented curve coverage*

Le *segmented curve coverage* sert à représenter des phénomènes qui varient le long de courbes. Il se base sur un objet de type *curve* (GM_Curve ; ISO 2003). Les méthodes d'interpolation sont linéaire, quadratique ou cubique.

2.2.2 Les images

Contrairement à la version précédente de l'*Abstract specification topic 6* (OGC 2000a), les images géographiques ne constituent plus une sous-classe de *coverage*. Elles ont été assimilées aux *grid coverages* et en particulier aux *rasters*. Néanmoins, certaines différences valent la peine d'être mentionnées (Donnay 2005).

⋆ La définition des cellules n'est pas la même (fig. 2.4).
En effet, une cellule d'un *grid* est définie entre quatre points de la grille. Ce n'est pas le cas des images pour lesquelles les cellules (pixels) sont définies autour des points. En fait, on peut voir les pixels comme les polygones de Thiessen construits sur les *grid points* - hormis ceux du bord. Le domaine spatial est donc plus étendu : il augmente de la résolution d'un pixel.

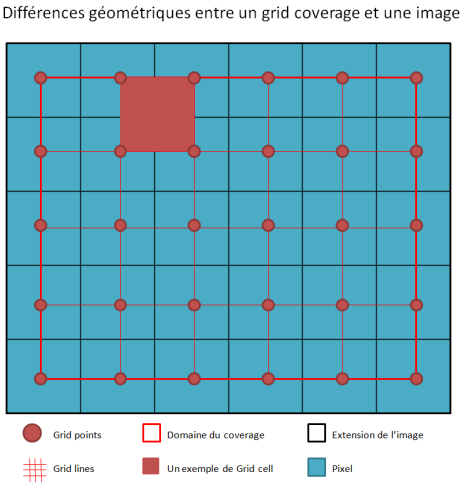


FIGURE 2.4 – Différences géométriques entre un *grid coverage* et une image

⋆ L'espace est discrétisé (fig. 2.5).
Un *grid coverage* est défini sur une collection de points alors que l'image repose sur un ensemble de surfaces. La valeur est identique sur tout le pixel comme s'il s'agissait d'un *continuous quadrilateral grid coverage* avec une interpolation au plus proche voisin. Cette discrétisation est souvent obligatoire, par exemple lors de la visualisation d'un modèle numérique de terrain sous forme d'une image ou

encore dans le cas des images aériennes pour lesquelles l'intégration d'une surface est obligatoire pour obtenir suffisamment d'énergie pour les photographies.

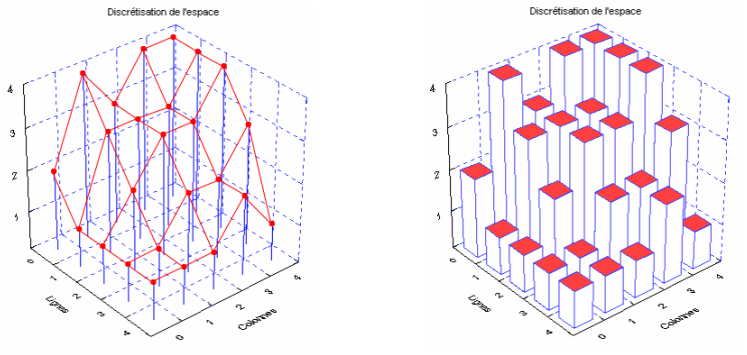


FIGURE 2.5 – La discrétisation de l'espace dans les images géographiques (Donnay 2005)

⋆ De nombreux traitements sont associés aux images géographiques.
Enfin, les images géographiques possèdent un panel de traitements spécifiques. Parmi ceux-ci, on peut retrouver le géoréférencement de photographies aériennes, l'algèbre de cartes, les traitements numériques d'images,... L'OGC a d'ailleurs rédigé trois spécifications sur le domaine des images : *Topic 7 : Earth Imagery* (OGC 2004), *Topic 15 : Image Exploitaion services* (OGC 2000b), *Topic 16 : Image Coordinate Transformation Services* (OGC 2000c).

L'image et le *continuous quadrilateral grid coverage*, pour autant qu'il soit *raster*, constituent les deux grandes classes d'information en mode maillé comme décrit au point 2.1.

Au-delà de la définition des *coverages*, l'OGC fournit également une description en GML (*Geographic Markup Langage* ; OGC 2007) de ceux-ci et un schéma de services web.

2.2.3 Les *coverage* et GML

Le type de base des *coverages* en GML est l'*AbstractCoverageType*, type qui sert à la définition de l'*AbstractDiscreteCoverageees* et de l'*Abstract Continuous Coverage*. Ces types sont abstraits et sont prévus pour être substitués. En particulier, L'OGC met à la disposition les types représentant les *discrete coverages* :

Le type *MultiPointCoverage* représente le *discrete point coverage*.
Le type *MultiCurveCoverage* représente le *discrete curve coverage*.
Le type *MultiSurfaceCoverage* représente le *discrete surface coverage*.
Le type *MultiSolidCoverage* représente le *discrete solid coverage*.
Le type *GridCoverage* représente le *discrete grid coverage* classique.
Le type *RectifiedGridCoverage* représente le *discrete grid coverage* possédant un système de référence externe.

2.2.4 Les services web de l'OGC

Le but des services web de l'OGC est de définir une interface standard et un ensemble d'opérations afin de faciliter l'accès à l'information géographique et pour permettre l'interopérabilité. Les services web de l'OGC sont répartis en trois : les *Web Map Services* (WMS, OGC 2006b), les *Web Feature Services* (WFS, OGC 2010c) et les *Web Coverage Services* (WCS, OGC 2010a). Comme son nom l'indique, les WMS sont des services dédiés aux cartes. Les WFS sont les services qui traitent des *features* telles que décrites par l'ISO (ISO 2003). Enfin, les WCS sont directement en rapport avec les *coverage* susmentionnés. Ce sont ceux qui nous intéressent.

Les WCS sont décrits via plusieurs documents. L'*OGC Web Services Commom Standard* (OWS, OGC 2010b) définit les aspects communs aux différents services de l'OGC (WMS, WFS et WCS). L'*OGC WCS 2.0 Interface Standard* (OGC 2010a) définit les opérations que doit supporter un serveur WCS alors que le *WCS Implementation Standard* (OGC 2008) décrit les modalités d'implémentation dudit serveur. Finalement, le *Web Coverage Processing Service(WCPS) Langage Interface Standard* (OGC 2009) décrit un langage permettant d'effectuer des traitements sur les *coverages* côté serveur avant de les récupérer. Le WCPS est une extension du WCS.

2.2.4.1 Le protocole WCS

L'OGC définit trois opérations que doit implémenter un serveur WCS. Il s'agit de :

- `GetCapabilities`
- `DescribeCoverage`
- `GetCoverage`

La requête `GetCapabilities` permet au client d'obtenir des informations à jour sur les *coverages* offerts par le serveur ainsi que des métadonnées de services. La réponse à cette requête fournit pour chaque *coverage* les informations suivantes :

- Un identifiant
- Le type de *coverage*
- Le cadre capable [1] dans le système WGS 84 en degrés décimaux (optionnel)
- Le rectangle capable dans le système de coordonnées propre au *coverage* (optionnel)
- Des métadonnées additionnelles (optionnel)

La requête `DescribeCoverage` se base sur les identifiants fournis par le `GetCapabilities` et retourne au client les informations suivantes :

- L'identifiant du *coverage* faisant l'objet de la description
- Une description du *coverage*
- Le domaine de définition du *coverage*
- Une description de la structure des attributs retournés par le *coverage*
- Des paramètres spécifiques au service
- La fonction d'interpolation (optionnel)
- Des métadonnées (optionnel)

La dernière requête, `GetCoverage`, permet de récupérer le *coverage* ou une partie de celui-ci : pour chaque *coverage*, le client peut préciser le sous-ensemble du

1. Le plus petit rectangle dont les côtés sont parallèles aux axes du système de référence et qui inclut tous les points de l'ensemble dont il est le cadre capable.

domaine qui l'intéresse. On parlera de *trimming* si on prend un ensemble plus petit mais de même dimensionnalité. En revanche, on parlera de *slicing* si on réduit la dimensionnalité (fig.2.6).

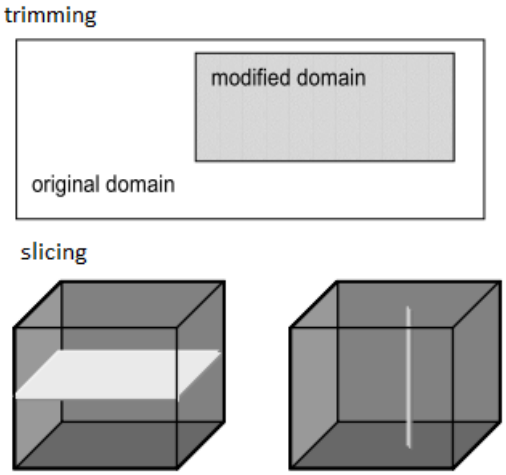


FIGURE 2.6 – Les opérations de *trimming* et *slicing* (d'après Baumann 2009b)

L'OGC propose trois modes pour délivrer une requête. Le premier repose sur la méthode `GET` du protocole HTTP (HyperText Transfort Protocol ; Kurose et Ross 2009). Il consiste à faire passer les paramètres de la requête sous forme de paires clé-valeur :

```
http://www.myserver.org:port/path?service=WCS&version=2.0
&request=DescribeCoverage&coverageid=C0002,C0003,C0004
```

Les deux autres méthodes consistent à fournir un document XML (eXtensible Markup Langage) reprenant les paramètres de la requête :

```
<DescribeCoverage>
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:wcs="http://www.opengis.net/wcs/2.0"
 xmlns:gml="http://www.opengis.net/gml/3.2"
 xsi:schemaLocation="http://schemas.opengis.net/wcs/2.0
  ../wcsAll.xsd"
 service="WCS" version="2.0.0">
 <wcs:CoverageId>C0001</wcs:CoverageId >
 <wcs:CoverageId>C0002</wcs:CoverageId >
 <wcs:CoverageId>C0003</wcs:CoverageId>
</DescribeCoverage>
```

La différence vient de la manière d'acheminer le document XML. La seconde méthode consiste à le délivrer dans le corps d'une requête HTTP via la méthode `POST`. Quant à la troisième méthode, elle repose sur le protocole SOAP (Simple Object Access Protocol) spécifié par le W3C (World Wide Web Consortium : http ://www.w3.org), où la requête est placée dans le corps du message SOAP.

Les réponses du serveur aux `GetCapabilities` et `DescribeCoverage` se font sous forme de fichiers GML (OGC 2007), éventuellement enfouis dans un message SOAP si c'est le mode de communication utilisé. Le choix est plus libre pour une requête `GetCoverage` où le résultat peut être présenté sous forme d'un fichier GML, d'un fichier binaire « bien connu » (comme le GeoTIFF) ou encore une combinaison des deux.

Dernières caractéristiques importantes : le protocole WCS gère l'absence de valeurs pour certains éléments des *coverages* et des messages d'erreurs sont spécifiés pour chacune des requêtes et des modes de communication.

2.2.4.2 Le langage WCPS

Au delà des opérations d'un serveur WCS, un serveur peut implémenter l'extension WCPS. Cette extension permet de réaliser des opérations sur les *coverage* avant de les récupérer. Contrairement au WCS qui définit un **protocole**, le WCPS définit un **langage**. Ce langage restreint quelque peu la notion de *coverage* comme suit :

- ⋆ Il ne s'adresse qu'aux *rasters*
- ⋆ Le type des attributs appartenant au *range* est limité :
  - seuls certains types de données sont possibles, comme le montre la figure 2.7
  - le *coverage* peut porter sur plusieurs attributs mais chacun d'eux ne peut avoir qu'un type atomique
  - tous les attributs d'un même *coverage* doivent être de même type

L'OGC considère néanmoins que cela recoupe la plupart des cas pratiques.

Range data type name	Meaning
boolean	Boolean
char	8-bit signed integer
unsigned char	8-bit unsigned integer
short	16-bit signed integer
unsigned short	16-bit unsigned integer
int	32-bit signed integer
unsigned int	32-bit unsigned integer
long	64-bit signed integer
unsigned long	64-bit unsigned integer
float	Single precision floating point number
double	Double precision floating point number
complex	Single precision complex number
complex2	Double precision complex number

FIGURE 2.7 – Les types de données autorisés par WCPS (OGC 2009)

La syntaxe du WCPS est la suivante :
Soient

$v_1, ..., v_n$	n itérateurs
$L_1, ..., L_n$	n listes de *coverages* avec$(n \geq 1)$
b	une expression booléenne (pouvant contenir plusieurs itérateurs)
P	une expression de traitement (pouvant contenir plusieurs itérateurs)

Alors une expression WCPS prend la forme :

```
for v1 in L1,
    ...
    vn in Ln
where b
return P
```

La multiplicité des itérateurs permet la construction de boucles imbriquées. À chaque itération, l'itérateur prend la valeur d'un *coverage* pour lequel le prédicat b est testé. S'il est négatif, l'itération continue. Sinon, le traitement P est appliqué, le résultat est ajouté à la pile des résultats et puis l'itération continue. Les listes de *coverages* doivent contenir les mêmes identifiants que ceux utilisés pour un `GetCoverage`.

Par exemple, si le serveur WCPS offre les *coverages* A, B et C, il est possible de les extraire au format TIFF de la manière suivante :

```
for $c in (A,B,C)
return tiff($c)
```

En outre, il est possible de faire référence à un seul attribut du *coverage*. Par exemple, si le serveur WCPS offre un *coverage* A qui possède un attribut nommé « red », la commande suivante renverra l'attribut « red » multiplié par deux sur l'ensemble du *coverage* A :

```
for $c in (A)
return $c.red * 2
```

Le WCPS est un langage riche permettant de réaliser la plupart des traitements attendus pour les images géographiques. Il supporte les opérateurs unaires et binaires classiques ($+, -, *, /$, `and`, `or`, `xor`, `not`, $<, \leq, >, \geq, =, \neq$), les changements de types et les arrondis, les fonctions trigonométriques (*sin*, *cos*, *tan*, *arcsin*, *arccos*, *arctan*), exponentielles (*exp*, *log*, *ln*) et autres fonctions usuelles (*pow*, *sqrt*, *abs*), l'*overlay*, des opérations sur le domaine spatial (*trimming*, *slicing*, extension du domaine, changement d'échelle et changement de référentiel) et des opérations de condensation afin d'extraire des statistiques (minimum, maximum, moyenne, total) ou de construire des fenêtres de convolution (Baumann 2010a).

Nous avons maintenant fait le tour de la normalisation des informations géographiques maillées par l'OGC. Dans la section suivante, nous allons nous intéresser aux systèmes opérationnels permettant de les traiter.

2.3 Les systèmes de gestion de l'information maillée : les solutions existantes

Après avoir longuement discuté de la normalisation de l'OGC dans le domaine des *coverages*, nous allons maintenant discuter des serveurs de données permettant de les gérer. Il faut malheureusement noter que la notion de *coverage* est relativement récente, elle n'a pris sa forme actuelle qu'en 2006 (OGC 2006a). De plus, avant sa dernière version (en 2010), le WCS ne gérait pas tous les *coverages* mais seulement les *rasters*. Si on rajoute à cela le fait que le langage WCPS ne gère encore que les *rasters*, il n'est pas étonnant que la gestion des *coverages* se résume à la gestion des *rasters* à l'heure actuelle.

C'est la raison pour laquelle ce mémoire s'intéresse aux serveurs de données *rasters*. Néanmoins, il est probable que des serveurs permettant la gestion des *coverages* apparaissent dans le futur.

En principe, stocker l'information géographique maillée dans une base de données est très simple (Donnay 2005). Les *rasters* sont stockés dans une table dont tous les attributs sauf un sont dédiés à la descriptions des métadonnées. Le dernier attribut est un *Binary Long OBject* (BLOB). Il s'agit d'un type particulier d'attributs permettant de stocker un fichier sous forme d'une chaîne binaire. Celui-ci contient la matrice des valeurs. À chaque *tuple*[2] de la table correspond donc un *raster*.

Cette approche est cependant théorique ; dès que la matrice des valeurs atteint une taille importante les performances de la base de données se dégradent (Donnay 2005). De plus, la base de données n'est pas compétente pour traiter l'information stockée dans le BLOB (Baumann et Widmann 1998). Il faut alors recourir à des stratégies telles que le découpage des *rasters* en tuiles rectangulaires (*tiles*), l'indexation spatiale et la définition de fonctions spécifiques. C'est cet enrichissement qui distingue la base données du serveur de données.

2.3.1 Oracle GeoRaster

Oracle GeoRaster (Oracle 2007) fait partie des produits Oracle Spatial (http ://www.oracle.com). Il s'agit d'un système de gestion de bases de données (SGBD) dédié aux *rasters*. Il est constitué des éléments suivants (fig. 2.8) :

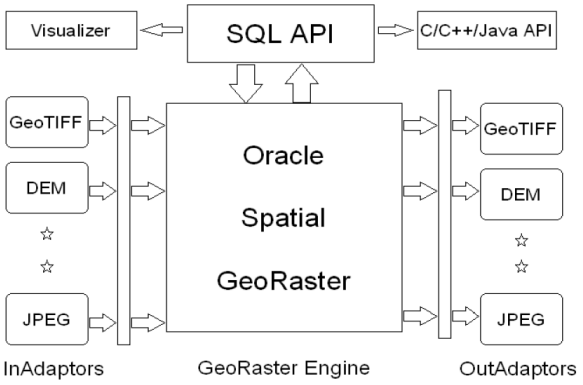


FIGURE 2.8 – Les composants d'Oracle GeoRaster (Oracle 2007)

Le GeoRaster Engine est le coeur d'Oracle GeoRaster. C'est lui qui gère la base de données, qui fournit la définition des *rasters* et les fonctionnalités pour les manipuler.

L'API SQL est une API[3] fournissant l'accès SQL standard aux données.

2. Grossièrement, un tuple correspond à une ligne dans une table

3. Une API (*Application Programming Interface*) est une bibliothèque reprennant un ensemble de fonctions permettant d'interagir avec un programme ; il s'agit d'un point d'accès vers le programme.

L'API C/C++/Java fournit un accès aux données via ces langages de programmation.

Les outils de visualisation permettent de visualiser les données au sein de la base de données. Un outil est fourni avec le SGBD mais d'autres outils plus évolués peuvent se connecter à la base de données.

Des filtres d'entrée et de sortie permettent la conversion depuis et vers des formats images et *rasters* (GeoTIFF, DEM, JPEG,...).

Les *rasters* sont vus comme des matrices de dimension deux (limitation de la version actuelle) potentiellement organisées en plusieurs couches d'informations. Des métadonnées sont disponibles pour le *raster* dans son ensemble (système de référence spatial et temporel notamment) et éventuellement pour chacune des couches.

Au niveau logico-physique (fig. 2.9), les *rasters* sont divisés en tuiles, indexées spatialement et stockées sous forme de BLOB. Trois tables sont utilisées pour la gestion des informations *raster*. La première table contient des informations « utilisateurs » ainsi qu'un objet `SDO_GEORASTER`. L'objet `SDO_GEORASTER` est documenté par une table du même nom reprenant les informations sur la géométrie de l'objet, les métadonnées globales ainsi qu'un identifiant permettant de retrouver les tuiles. Celles-ci sont stockées dans une troisième table : `SDO_RASTER` qui reprend l'identifiant susmentionné, le cadre capable de la tuile et la valeur de celle-ci sous forme de BLOB.

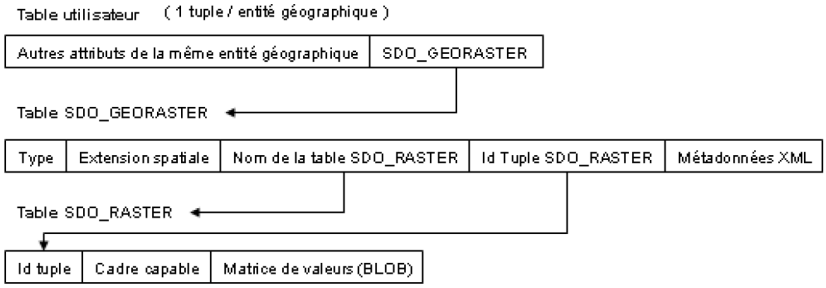


FIGURE 2.9 – Structure des tables dans Oracle GeoRaster (Donnay 2005 d'après Oracle 2003)

Les opérations (requêtes, mises à jour, traitements) disponibles pour les *rasters* se font via les fonctions fournies par GeoRaster.

2.3.2 La solution d'ESRI

ESRI (http ://www.esri.com) offre la possibilité de « rasteriser » une base de données via son module Raster Data d'arcSDE (ESRI 2005). L'architecture repose sur sept tables (fig. 2.10). La table spatialisée (table « utilisateur ») se voit ajouter deux colonnes : une colonne `raster` qui contiendra la clé externe vers le *raster* et une colonne `footprint` qui contiendra une clé externe vers le cadre capable du *raster*. La table spatialisée est référencée dans la métatable `SDE_raster_column` qui contiendra parmi ses métadonnées le système de référence. La clé externe `raster` permet d'identifier, d'une part, un *tuple* dans la

table `SDE_ras_n` qui représente le *raster* et, d'autre part, plusieurs *tuples* représentant les différentes couches dans la table `SDE_bnd_n`. Cette dernière table a comme clé primaire `rasterband_id` qui permet d'identifier à quelle bande appartiennent les tuiles stockées dans la table `SDE_blk_n`. Les métadonnées des bandes sont stockées dans la table `SDE_aux_n`. Enfin, le cadre capable est stocké dans les tables `F_tables` et `S_tables`.

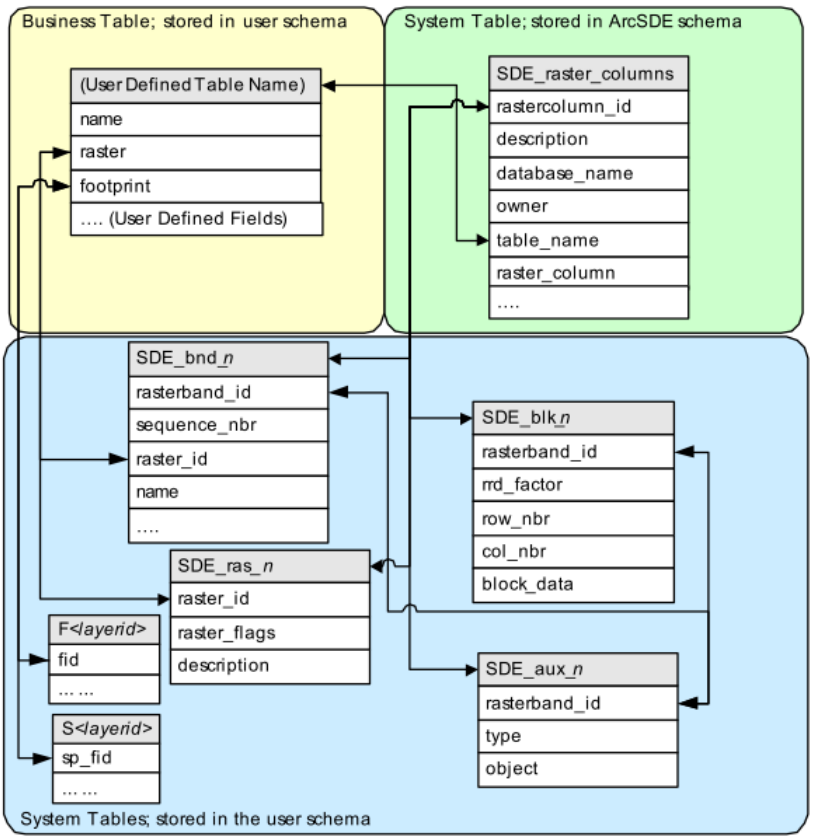


FIGURE 2.10 – Structure des tables d'une « rasterisation » par arcSDE (ESRI 2005)

Aucun langage ou ensemble de fonctions de manipulation ne sont prévus par arcSDE ; les traitements sont censés être exécutés via arcGIS comme s'il s'agissait de *rasters* stockés sur le disque dur.

2.3.3 PostGIS Raster

PostGIS Raster (anciennement WKTRaster : http ://trac.osgeo.org/postgis/wiki/WKTRaster) est une extension de PostGIS (http ://postgis.refractions.net), l'extension spatiale de PostgreSQL (www.postgresql.org) un SGBD objet-relationnel *open source*. PostGIS Raster, comme son nom l'indique, prend en charge les informations *raster*. Cette extension est toujours en développement et bon nombre de fonctionnalités ne sont pas encore implémentées. Néanmoins, il est déjà disponible sous

forme d'un module séparé. Il devrait être intégré à PostGIS dans sa prochaine version (2.0) attendue pour l'automne 2011.

PostGIS Raster implémente le type complexe `Raster` qui peut servir à la fois de représentation pour un *raster* complet ou pour ses tuiles. Ce type gère les *raster* de dimension deux mais pouvant posséder plusieurs bandes. C'est lui qui contient toutes les métadonnées (système de référence, taille, résolution,...) et la matrice de valeurs. Outre la table « rasterisée », il existe une table `raster_columns` résumant les informations importantes de l'ensemble des tables spatiales. Cependant, il s'agit bien là d'un résumé centralisant les informations ; le type `Raster` contient toutes ces informations. La spatialisation des tables est donc simple et similaire à celle de PostGIS.

Une table « rasterisée » contient une colonne de type `Raster`. Le choix de la signification de la table est laissé à l'utilisateur. Il peut s'agir d'une table :

- ⋆ reprenant un ensemble d'images non subdivisées en tuiles
- ⋆ décrivant une seule image où chaque tuple définit une tuile
- ⋆ décrivant un « *raster coverage* » au sens de PostGIS : une collection de tuiles non-contiguës et de tailles différentes.

À l'instar de PostGIS classique, PostGIS Raster définit un grand nombre de fonctions permettant de traiter les *rasters* directement dans les requêtes SQL. Ce sont celles-ci qui ne sont pas encore toutes implémentées. De plus, PostGIS Raster est conçu pour pouvoir travailler simultanément sur les *rasters* et sur les informations vectorielles. Dernière particularité de PostGIS Raster, il permet de travailler sur des *rasters* dont la matrice de valeurs est stockée dans un fichier sur le disque dur. Il intègre alors les métadonnées afin de pouvoir bénéficier du langage de requête et des indexations spatiales.

2.3.4 TerraLib

TerraLib (www.terralib.org) est un projet *open source* visant à fournir un ensemble de fonctionnalités du domaine des SIG et de l'analyse spatiale via un API C++. Il est interopérable avec plusieurs bases de données (Oracle Spatial, PostgreSQL, PostGIS) qu'il spatialise différemment pour tirer au mieux parti des fonctionnalités de chacune d'entre elles. Néanmoins, la gestion des *rasters* est commune. Les *rasters* sont de dimensions deux et composés de plusieurs couches. Quatre tables servent à la gestion des *rasters* (fig. 2.11). La table `Raster` stocke les métadonnées de l'ensemble du *raster*, telles que la résolution spatiale, le nombre de bandes, le cadre capable et la taille du *raster*. La table `Raster_metadata` stocke les informations de chaque couche : le minimum et maximum, le type de données, la compression utilisée,... La table `raster_table` est la table qui contient les tuiles des rasters dans des BLOB. Enfin, la table `te_lut_table` est une table servant pour la conversion de couleurs indexées.

TerraLib dédie trois classes de son API à la gestion les *rasters* :

TeRaster est une classe générique permettant de gérer les *rasters*.

TeRasterParams est une classe permettant d'accéder aux paramètres des *rasters* (système de référence, taille du *raster*, nombre de bandes,...)

TeDecode est une classe permettant la gestion des *rasters* au format binaire. Elle permet donc d'importer et d'exporter depuis la base de données.

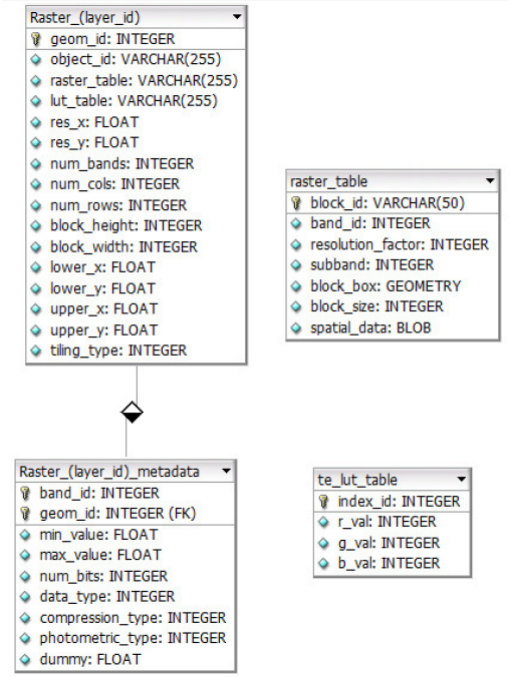


FIGURE 2.11 – Structure des tables dans TerraLib

Finalement, TerraLib met à disposition des fonctions spatiales dont une partie permet d'effectuer des opérations sur les *rasters*.

2.3.5 Le projet Sequoia 2000

Le projet Sequoia 2000 est un vieux projet (Stonebraker 1994) qui voulait offrir un meilleur environnement informatique aux chercheurs sur le changement climatique. Les *rasters* représentant la grande part des informations en terme de masse, le projet Sequoia 2000 s'est vu obligé de les intégrer, parmi d'autres informations géographiques, dans son système d'information. Celui-ci reposait sur PostgreSQL (www.postgresql.org) pour lequel ils ont dû développer des stratégies de découpage en tuile et des index spatiaux.

Il s'agit cependant d'un projet ciblé qui n'a pas été conçu pour être réutilisé dans d'autres domaines.

2.3.6 SciDB

Contrairement au projet Sequoia 2000, le projet SciDB (http ://www.scidb.org) est un projet récent (Becla et Lim 2008). Il s'articule sur un modèle de données basé sur des tableaux (dans le sens informatique du terme) qui peuvent représenter les *rasters*. Il n'y a cependant pas de mentions d'aspect géographique.

La documentation est encore assez éparse, notamment sur le fonctionnement interne de SciDB mais il est clair que celui-ci ne repose pas sur un SGBD « classique ». En particulier, il ne respecte pas le schéma classique des transactions ACID (Atomicity, Consistency, Isolation, Durability) (Gardarin 2003).

2.3.7 MySQL

MySQL Server (http ://dev.mysql.com) possède bien une extension spatiale mais celle-ci ne gère pas le *raster*.

2.3.8 Rasdaman

Les quelques solutions passées en revue ont chacune leurs limitations et inconvénients. GeoRaster est certainement la solution la plus aboutie mais elle est commerciale. TerraLib semble être le projet *open source* le plus développé mais il ne constitue pas un SGBD en soi. Le projet Sequoia 2000 n'est pas prévu pour être transposé. La « rasterisation » par arcSDE d'ESRI permet de stocker les *rasters* mais ne permet guère plus. Quant à SciDB et PostGIS Raster, ils sont prometteurs mais toujours en développement...

Il reste néanmoins une solution logicielle : Rasdaman se présente comme une extension des SGBD relationnels permettant de stocker, récupérer et traiter les informations *rasters* de taille et dimensionnalité illimitées via un langage de requête étendu depuis le SQL et qui pourrait servir de serveur WCS et WCPS.

Chapitre 3

Objectif et méthodologie

Dans cette partie, nous examinerons l'objectif du mémoire. Cet objectif sera ensuite reformulé sous la forme d'une hypothèse sur laquelle ce mémoire devra statuer. En dernier lieu, nous développerons la démarche utilisée ainsi que l'ensemble des critères permettant de confronter l'hypothèse.

3.1 L'hypothèse

L'objectif de ce mémoire est de tester les capacités de Rasdaman comme serveur de données géographiques maillées et notamment son utilisation dans le projet CARE GITAN de l'ULg. Ce projet tend à rassembler toutes les informations *rasters* de l'ULg dans un même serveur afin qu'elles soient disponibles pour tous les chercheurs de l'ULg.

D'une manière générale, un serveur de données est un logiciel permettant de stocker et organiser l'information ainsi qu'offrant des facilités pour la retrouver. Ce logiciel est calibré pour les données dont il est responsable. Il est efficace et dispose d'une « compréhension » de celles-ci ; il ne se contente pas de renvoyer des fichiers dont il ne comprend pas le sens. De tels serveurs offrent donc généralement la possibilité d'effectuer des traitements afin de ne transmettre que la partie pertinente des informations et ainsi ne pas encombrer le réseau. La délimitation précise des capacités attendues du serveur de données **géographiques** maillées est matérialisée par les critères de validation qui vont suivre (3.3).

Au final, l'hypothèse sur laquelle ce mémoire se base est la suivante :

> « Rasdaman, à lui seul, est capable de faire office de serveur de données géographiques maillées, notamment dans le cadre du projet CARE GITAN de l'ULg ».

3.2 La méthodologie

La confrontation de l'hypothèse se fera en quatre étapes. La première étape consiste à dresser une liste des critères que doit remplir Rasdaman afin de valider l'hypothèse. Cette liste de critères est organisée en trois parties : fonctionnalités, données et, traitements et requêtes. Ceux-ci sont repris à la section 3.3. Parmi

ces critères, certains sont obligatoires et d'autres optionnels. Ces derniers n'ont pas d'impact sur la validation ou non de l'hypothèse mais représentent des informations intéressantes.

La seconde étape (chapitre 4) consiste à confronter ces critères de validation à la documentation logicielle et scientifique de Rasdaman afin de voir ce qu'offre le logiciel. Il s'agit donc de trier parmi les critères ceux qui sont exclus d'office par Rasdaman. Cette première confrontation conclura la présentation de Rasdaman (section 4.7). Cette étape devrait faire le point sur l'aspect fonctionnalité de la liste de critères.

La troisième étape (chapitre 5) consiste à tester les critères restants par prototypages. Cette étape s'effectuera en deux temps :

1. **Le prototypage des données** (section 5.2). Il s'agit de sélectionner un jeu de données rapport avec les critères de validation des données et des traitements. Cela implique également de définir comment organiser les données au sein du serveur. Ensuite, il faut procéder au chargement des données et en mesurer les performances.
2. **Le prototypage des traitements et des requêtes** (section 5.3). Dans un premier temps, il faut les formuler dans le langage d'interrogation du serveur et les exécuter. Après cela il faut les valider en comparant le résultat via des logiciels tiers ; dans le présent mémoire, le logiciel Idrisi de Clark lab (www.clarklabs.org) et la bibliothèque GDAL (*Geospatial Data Abstract Library*, www.gdal.org). GDAL permettra de valider les traitements et requêtes touchant aux statistiques et aux métadonnées. Idrisi validera le reste. Pour ce faire, les mêmes traitements seront réalisés par celui-ci. Les résultats seront alors comparés cellule à cellule. Enfin, il faudra évaluer les performances des requêtes et des traitements. La manière de réaliser ces mesures dépendra des possibilités offertes par Rasdaman. Ce point sera rediscuté lors du prototypage (section 5.1.2).

La troisième étape se termine sur une deuxième confrontation partielle avec les critères de validation. Il s'agit de faire le point sur ce que le prototype a mis en évidence.

Dans la quatrième et dernière étape (chapitre 6), on revient sur l'hypothèse. On passe la liste des critères en revue et on statue sur chacun d'eux. Cette étape sera d'autant plus rapide que nous avons réalisé des confrontations partielles. Pour pouvoir valider l'hypothèse, tous les critères obligatoires doivent être satisfaits. Sinon, l'hypothèse doit être rejetée. Finalement, si tel est le cas, quelques pistes seront proposées pour contourner les obstacles soulevés.

La méthode utilisée pourrait permettre de statuer rapidement sur l'hypothèse s'il advenait qu'un critère ne soit pas rempli. Néanmoins, tous les critères seront examinés afin de faire le point complet sur Rasdaman. Cette approche permettrait aussi d'envisager une solution surmontant tous les obstacles rencontrés, s'il advenait que l'hypothèse soit rejetée.

3.3 Les critères de validation

3.3.1 Les fonctionnalités

Un serveur devant valider l'hypothèse de ce mémoire devrait supporter les fonctionnalités suivantes :

- ⋆ Offrir des mécanismes d'insertion et d'extraction des données depuis et vers les formats de données *rasters* et images les plus connus.
- ⋆ Offrir un mécanisme permettant d'effectuer des requêtes et des traitements.
- ⋆ Offrir des capacités de création de catalogues de données.
- ⋆ Offrir des méthodes d'indexation pour un accès rapide à l'information.
- ⋆ Éventuellement, offrir la possibilité de créer un serveur web de données *rasters*.
- ⋆ Éventuellement, offrir des outils facilitant la conception d'un serveur de données.

3.3.2 Les données

En ce qui concerne les données, le serveur de données doit répondre aux critères suivants :

- ⋆ Gérer les métadonnées générales et notamment les métadonnées spatiales (système de référence, résolutions,...).
- ⋆ Gérer différents types de données :
  - Images géographiques 2D binaires, nominales, quantitatives.
  - *rasters* 2D et 3D : modèles numériques de terrain, champs de températures,...
  - Collection d'images géographiques : multi-spectrales, diachroniques.
  - Éventuellement, intégrations de données vectorielles.
- ⋆ Éventuellement, gérer des métadonnées particulières :
  - Couleurs indexées.
  - Légendes.
- ⋆ Pour autant que possible, garder la gestion interne des *rasters* la plus simple possible.

3.3.3 Les traitements et les requêtes

Le serveur de données devrait répondre aux critères suivants en terme de traitements et de requêtes :

- ⋆ Offrir un mécanisme permettant d'effectuer des requêtes :
  - Sur les métadonnées.
  - Éventuellement sur les *rasters*.
- ⋆ Permettre les traitements courants de l'algèbre de carte et du traitement d'image :
  - Calculer des statistiques sur l'image.
  - Jouer sur la résolution spatiale.
  - Exécuter des opérations binaires (+, *, -, /, `and`, `or`, `xor`,...)
  - Gérer les projections.
  - Créer des mosaïques.
  - Réaliser des masquages et des reclassifications.
  - Calculer des histogrammes et des améliorations de contrastes.
  - Appliquer des fenêtres de convolution.
  - Créer des compositions colorées.

Comme expliqué précédemment, pour valider un traitement, il faut qu'il soit :

- ⋆ Réalisable par Rasdaman.
- ⋆ Juste : il doit être validé par un programme tiers.
- ⋆ Performant.

La notion de performance est délicate à évaluer. Ici, nous retiendrons comme critère le temps. Il est cependant malaisé de déterminer le temps maximum que peut prendre une requête. Il dépend de la complexité de celle-ci, de la taille des données qu'elle doit traiter et de la machine qui l'exécute. C'est pourquoi les performances mesurées sont principalement indicatives. Des requêtes très coûteuses en temps (dépassant la minute) peuvent néanmoins être jugées non performantes.

Chapitre 4

Rasdaman

Dans cette partie, Rasdaman est présenté. Une grande part des informations vient de la documentation de Rasdaman (parfois corrigée). La littérature scientifique autour de Rasdaman a également fourni des informations précieuses. Enfin, la section concernant la gestion interne des données (4.5) n'a pu être alimentée que par *retro-engineering*.

4.1 Une brève présentation de Rasdaman

Rasdaman (*RASter DAta MANager*, www.rasdaman.org) se présente comme une extension des SGBD relationnels permettant de stocker, récupérer et traiter les informations *raster* de taille et dimensionnalité illimitées via un langage de requête étendu depuis le SQL et qui pourrait servir de serveur WCS et WCPS.

Rasdaman est un projet qui remonte à la fin des années 1990 (Ritsch *et al.* 1997) et qui visait à offrir une gestion des *rasters* basée sur une algèbre spécifique aux données multidimensionnelles discrètes et indépendante d'un domaine d'application. En 2009, la version 8.0 de Rasdaman devient accessible en *open source* sous le nom de *Rasdaman community*. Néanmoins, une version commerciale est toujours disponible et propose des fonctionnalités supplémentaires (fig. 4.1). Parmi celles-ci, on note des améliorateurs de performances ainsi que les *CRS transformations* qui signifient les changements de systèmes de coordonnées. La version *open source* ne gère donc pas les projections cartographiques. Par la suite, quand nous parlerons de Rasdaman, nous ferons référence à la version *open source*.

Le responsable du projet, Mr. Baumann, est professeur de sciences informatiques à l'université de Brême. Il est aussi impliqué à l'OGC, notamment dans la branche des WCPS (Baumann 2010a).

Rasdaman est prévu pour fonctionner sous linux et ses modules sont appelés depuis la console.

rasdaman version:	community	enterprise	
Main functionality:			
rasql	X	X	every query will run on both versions
C++ API	X	X	C++ clients using community functionality will run on both versions
Java API	X	X	Java clients using community functionality will run on both versions
Extra functionality:			
dynamic data definition	X	X	new cell, array, and collection types can be added at runtime
CRS transformation		X	
Storage:			
tiling	X	X	
tiling policies	X	X	
storage layout language	X	X	
spatial indexes	X	X	
input/output data formats	X	X	
compression		X	
tape archive integration		X	
Performance accelerators:			
parallel server processes	X	X	
distributed processing		X	
heuristic optimization		X	
just-in-time compilation		X	
GPU query evaluation		X	
server management:			
via command line	X	X	
via Web browser		X	
OGC interfaces:			
WCS, WCPS, WPS	X	X	
WMS		X	waiting for volunteers to integrate it with rasdaman community
WMS client		X	...but any other (such as OpenLayers) will do as well, of course
Base DBMSs supported:			
PostgreSQL	X	X	
MySQL		X	
Oracle		X	
IBM Informix		X	
IBM DB2		X	
file system		X	

FIGURE 4.1 – Les différences entre la version open source et la version commerciale de Rasdaman

4.2 Structure et architecture

Rasdaman s'inscrit dans le paradigme client-serveur où il se présente comme un *middleware* (fig. 4.2) ; une couche logicielle se plaçant entre le SGBD et le client qui offre des fonctionnalités supplémentaires au serveur.

Rasdaman est constitué des composants suivants (fig. 4.2). Entre parenthèses se trouvent les modules correspondants appelables depuis la console linux :

Un médiateur (`rasmgr`) dont le rôle est de recevoir les requêtes des clients et de les répartir sur les différents serveurs (`rasserver`).

Des serveurs (`rasserver`) qui traitent les requêtes.

Une interface administrative (`rascontrol` et `raspasswd`) qui sert à l'administration du SGBD.

Un langage d'interrogation (`rasql` et `rasdl`) qui sert à la manipulation des données.

Un visualisateur (`Rview`).

Des filtres pour importer et exporter les données.

Des API C/C++ et Java qui permettent le développement d'applications.

Les éléments principaux sont maintenant passés en revue. Les langages de manipulation seront détaillés plus bas (section 4.4).

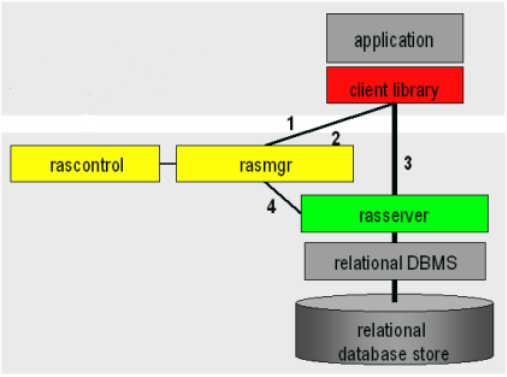


FIGURE 4.2 – Architectures de Rasdaman (Baumann 2009&)

4.2.1 Les serveurs

Les serveurs sont le coeur de Rasdaman (fig. 4.3). Le médiateur leur transmet les requêtes des clients qu'ils décodent en reformant un arbre de requête. L'arbre est ensuite optimisé pour parcourir et traiter le moins de tuiles possibles et dans l'ordre le plus rapide (Furtado *et al.* 1997). Les serveurs construisent alors un flux de tuiles : une tuile est récupérée et traitée avant de passer à la tuile suivante. Ainsi, il est possible de traiter des fichiers dépassant de loin la taille de la mémoire vive de l'ordinateur.

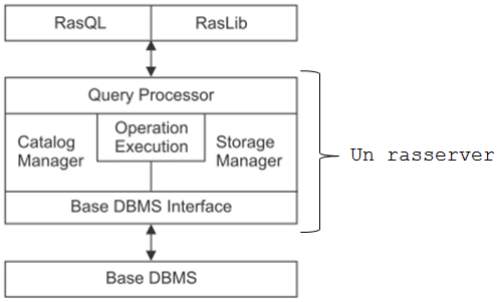


FIGURE 4.3 – La place des `rasservers` dans l'architecture (d'après Baumann et Widmann 1998)

Vu que la construction des *rasters* et le découpage en tuiles passent par le langage de manipulation qui est géré par le `rasserver`, c'est donc lui aussi qui gère ces éléments au final. En particulier, le *Catalog Manager* est en rapport avec la construction des *rasters* et le *Storage Manager* l'est avec le découpage en tuiles.

4.2.2 Le médiateur

Le médiateur est un processus tournant en fond. C'est à lui que le client se connecte. Le médiateur peut alors rediriger le client vers un serveur libre.

Les connexions sont de types TPC/IP et trois protocoles sont possibles pour dialoguer entre client et serveur, même si les deux premiers sont dépréciés. Le protocole RPC (Remote Processing Call) sert toujours pour la communication avec le visualisateur `Rview`. Le protocole HTTP est lui aussi déprécié. Le protocole à utiliser est donc le protocole maison : Rasdaman Network Protocol (RNP).

C'est aussi le rôle du médiateur d'instancier les différents serveurs. Les paramètres de ceux-ci peuvent être configurés statiquement (*i.e.* lorsque le médiateur est éteint) via un fichier de configuration ou dynamiquement en passant par le module `rascontrol`.

4.2.3 Les outils d'administration

Le module `rascontrol` est le point d'accès administrateur au médiateur et aux serveurs. Il permet d'instancier dynamiquement des serveurs et de les configurer, de définir les hôtes chez lesquels opèrent les serveurs ainsi que les bases de données sur lesquels Rasdaman opère. C'est aussi par lui qu'il faut passer pour définir de nouveaux utilisateurs et leurs droits.

Quant au module `raspasswd`, il permet aux utilisateurs de changer leur mot de passe. Deux utilisateurs de Rasdaman sont créés lors de l'installation : `rasguest` et `rasadmin`. Les mots de passe sont identiques aux noms des utilisateurs. Le premier ne possède que le droit d'extraire des données alors que le second a tous les droits. Ces utilisateurs sont en lien exclusivement avec Rasdaman. En interne, Rasdaman se connecte au SGBD sous le nom de l'utilisateur `rasdaman`.

4.2.4 Les filtres

La manière la plus simple de fournir et d'extraire des informations de Rasdaman est de passer par une série de filtres qui sont repris à la figure 4.4. Le format DEM correspond à un fichier ASCII où chaque ligne comprend les coordonnées des points (x, y et z) séparés par des espaces. Ces filtres font partie intégrante du langage de requête : on les invoque via des fonctions.

Image format	rasql conversion function	Dimension
JPEG	`jpeg(), inv_jpeg()`	2
PNG	`png(), inv_png()`	2
TIFF	`tiff(), inv_tiff()`	2
BMP	`bmp(), inv_bmp()`	2
VFF	`vff(), inv_vff()`	3
HDF 4	`hdf(), inv_hdf()`	2,3
DEM[8]	`dem(), inv_dem()`	2
TOR	`tor(), inv_tor()`	2

FIGURE 4.4 – Les formats d'entrées et sorties (Baumann 2009b)

Il faut noter que les aspects spatiaux sont perdus lors des conversions. Par exemple, si une image GeoTIFF contenant des informations sur le référentiel spatial est insérée dans la base de données, les informations spatiales sont perdues. Ce point sera largement débattu par la suite.

En outre, Rasdaman est prévu pour la gestion du *raster* et des images. Il ne sait pas gérer les autres formes de *coverages* définit par l'OGC.

4.2.5 Conclusion sur l'architecture et la structure de Rasdaman

Rasdaman est donc un *middleware* dont le but est de « rasteriser » la base de données sur laquelle il travaille tout en fournissant un ensemble de fonctionnalités pour gérer les informations *rasters*. Néanmoins, Rasdaman ne fait office de serveur de données **que** pour les *rasters* (fig. 4.5). Il faut donc stocker les métadonnées séparément (mais dans la même base de données) et Rasdaman ne sait pas les intégrer dans son langage de requête. Il s'agit donc d'une variante d'un modèle de gestion hybride des informations.

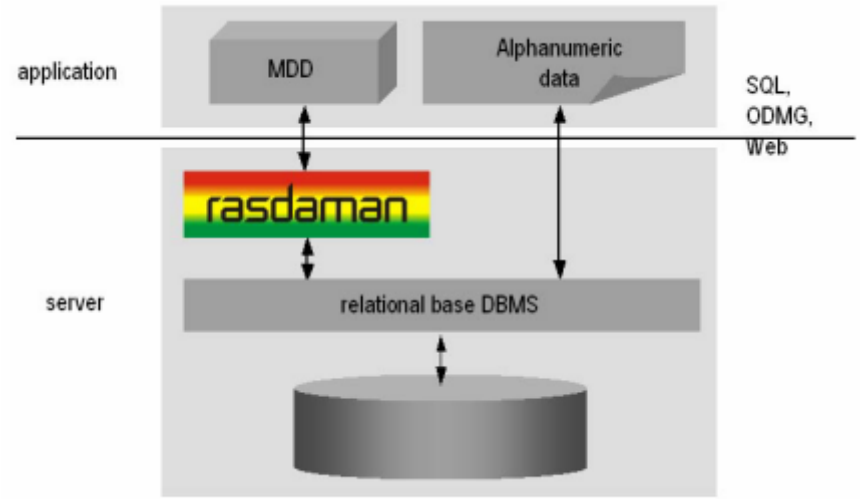


FIGURE 4.5 – L'architecture hybride de Rasdaman (Baumann 2009b)

4.3 L'approche conceptuelle de Rasdaman pour la gestion des *rasters*

L'approche de Rasdaman pour traiter les *rasters* est présentée à la figure 4.6. Les *rasters* sont vus comme un tableau (dans le sens informatique du terme) multidimensionnel appelé MDD (*Multidimensional Discrete Data*). Rasdaman ne fait donc pas de distinctions entre *rasters* et images et rassemblent les deux sous le vocable de MDD.

Un MDD possède d'abord un domaine spatial. Ce domaine définit la dimensionnalité du MDD ainsi que son extension dans chaque dimension. L'extension est définie comme un intervalle d'entiers mais qui ne doit pas nécessairement commencer à zéro. Il n'y a pas de restrictions sur la dimensionnalité excepté le fait qu'il doit s'agir d'un nombre entier supérieur à zéro. Un vecteur de coordonnées fait référence à une cellule qui contient la valeur du phénomène.

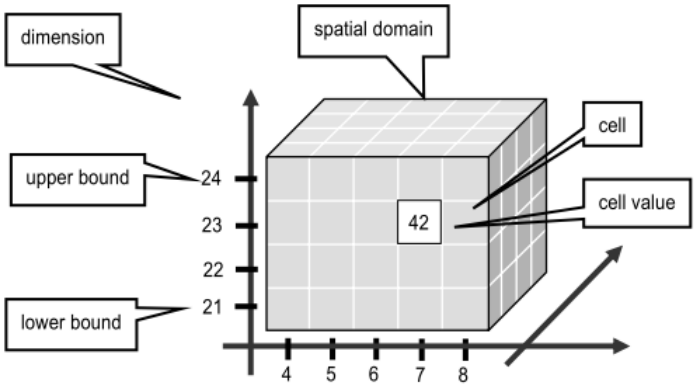


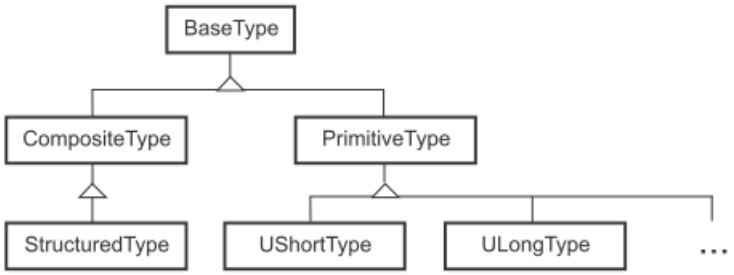
FIGURE 4.6 – Modèle conceptuel du *raster* dans Rasdaman (Baumann 2009b)

4.3.1 Les cellules

Le type de la cellule, appelé `BaseType`, peut être simple ou composé (fig. 4.7). Les types simples constituent les primitives et sont illustrés par la figure 4.7b. On retrouve les types habituels : booléen, codage sur 8 et 16 bits signés ou non, réels à simple (32 bits) et double (64 bits) précision, ainsi que deux types servant à représenter les nombres complexes en simple et double précision.

Dans le cas d'un type composé, la cellule ne renverra plus une valeur mais un vecteur de valeurs. En informatique, on parlera de **structure**. Contrairement au WCPS, Rasdaman n'impose pas la limitation que toutes les données d'un type composé aient la même primitive. Ainsi, il est possible de représenter un *raster* dont la première composante représenterait un modèle numérique de terrain codé en réel sur 64 bits, la deuxième représenterait un masque d'altitude codé sur 1 bit et la troisième représenterait l'occupation du sol codée sur 8 bits non signés. En revanche, Rasdaman impose que les différentes composantes aient la même extension spatiale. Ainsi, il n'est pas possible de représenter par un seul MDD un ensemble d'images satellites représentant une même scène à des résolutions spatiales différentes.

On peut noter que la plupart des types décrits par l'OGC pour le WCPS sont repris ici. Les noms ne sont, cependant pas forcément les mêmes. Seuls les entiers codés sur 64 bits (signés ou non) ne sont pas présents dans Rasdaman.



(a) Hiérarchie des types de données dans Rasdaman (Baumann et Widman 1998)

rasdl name	size	description
`octet`	8 bit	signed integer
`char`	8 bit	unsigned integer
`short`	16 bit	signed integer
`unsigned short`	16 bit	unsigned integer
`long`	32 bit	signed integer
`unsigned long`	32 bit	unsigned integer
`float`	32 bit	single precision floating point
`double`	64 bit	double precision floating point
`complex`	64 bit	single precision complex
`complexd`	128 bit	double precision complex
`boolean`	1 bit[2]	`true` (nonzero value), `false` (zero value)

(b) Primitives de Rasdaman (Baumann 2009b)

FIGURE 4.7 – Les types de données dans Rasdaman

4.3.2 Les collections

Enfin, les MDD sont organisés en collections. Une collection peut accueillir des MDD de même type. Quand on parle du type de MDD, on parle du type de cellules qu'il contient mais également de la dimensionnalité et de ses contraintes sur l'extension spatiale. La collection dans Rasdaman est l'équivalent de la relation en algèbre relationnelle. À part cela, elle n'a pas de signification particulière. Une collection peut contenir des images qui sont en rapport avec une même scène à une même époque dans des bandes spectrales différentes, une analyse diachronique d'une même scène ou un groupe d'images d'un même satellite représentant des scènes différentes à des époques différentes.

4.3.3 Dernières remarques

Rasdaman est très souple dans la définition de ses données. Ceci peut s'illustrer par l'exemple d'un groupe d'images mono-spectrales diachroniques. On peut

stocker les différentes images au sein d'une même collection en plusieurs MDD (1 époque = 1 MDD, 1 seule composante, 2D). Il est possible de stocker les images comme un seul MDD avec type composé (1 MDD, 1 époque = 1 composante, 2D). Enfin, on peut créer un MDD 3D : 2D spatiales et 1D temporelle (1 seul MDD, type primitif, 3D). Le débat peut être étendu à beaucoup d'autres cas, notamment celui des images multi-spectrales diachroniques.

Dans la suite de ce travail, l'abus de langage « MDD primitif » signifiera MDD dont les cellules ont un type primitif. De même, « MDD composé » signifiera l'inverse : un MDD dont les cellules ont un type composé.

4.4 Le langage de Rasdaman

Afin de manipuler les données stockées en son sein, Rasdaman offre un langage de requêtes qui est une extension du langage SQL : le Rasql (*Rasdaman query langage*). Celui-ci se divise en deux parties :

Rasdl le langage de définition des données (*Rasdaman definition langage*). C'est lui qui permet, notamment, de définir les types composés.

Rasml le langage de manipulation des données (*Rasdaman manipulation langage*). C'est lui qui permet de faire les requêtes classiques : `select`, `insert`, `update`, `delete`.

Il faut cependant noter une confusion potentielle dans les termes. Rasdl, le langage, est géré par le module `rasdl`. Alors que le langage Rasml est utilisé via le module `rasql`. Nous allons maintenant passer ces deux langages en revue.

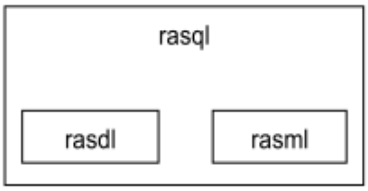


FIGURE 4.8 – Le langage de requête de Rasdaman (Baumann 2009)

Les conventions suivantes seront utilisées pour décrire la syntaxe du langage de requête :

- ⋆ Les éléments en caractères gras sont des mots réservés ; il ne faut donc pas les changer.
- ⋆ Les éléments commençant par un « £ » sont des éléments devant être adaptés.
- ⋆ Les crochets ([]) précédés d'un « £ » indiquent les éléments optionnels.
- ⋆ Le symbole * signifie que les éléments **peuvent** être présents et répétés plusieurs fois ($n \geq 0$).
- ⋆ Le symbole + signifie que les éléments **doivent** être présents et peuvent être répétés ($n > 0$).

4.4.1 Rasdl

Rasdl va permettre de définir les types de cellules, les types de MDD et les types d'ensembles.

4.4.1.1 Les types composés

La syntaxe utilisée pour définir les types composés est proche de celle du langage C++ pour définir des structures. Le nom du type composé succède au mot clé `struct`, suivent alors les composantes entre accolades. Celles-ci sont décrites par leur type primitif suivi d'un nom. Plusieurs composantes peuvent avoir le même type, auquel cas il suffit de séparer leur nom par des virgules. Sinon, les composantes sont séparées par les points virgules.

```
struct £nom_de_la_structure
    {£type_primitif_1 £nom_de_la_composante_1
        £[, £nom_de_la_composante_2]*;
     £[£type_primitif_n £nom_de_la_composante_z
        £[, £nom_de_la_composante_w]*;]*
    };
```

Par exemple, si on veut décrire une image RVB (Rouge, Vert, Bleu) dont chaque composante offre 256 niveaux (*i.e.* les images qu'on rencontre couramment), la structure serait la suivante :

```
struct RGB {char red, green, blue;};
```

Et si on veut décrire la structure mentionnée précédemment à la section 4.3.1 (avec le modèle numérique de terrain, le masque binaire et l'occupation du sol) :

```
struct MNT_M_OS {double MNT; boolean  masque ;
    char occupation_du_sol; };
```

4.4.1.2 Les types de MDD

Un type de MDD définit le nom du type, le domaine spatial que pourront prendre ses instances et le type des cellules. La syntaxe est la suivante :

```
typedef marray <£nom_de_la_structure, £domaine_spatial>
    £nom_du_type_de_MDD
```

Rasdaman offre une bonne flexibilité sur la manière de définir le domaine spatial. Ainsi on peut le définir des façons suivantes :

explicitement $[-1:1, -1:1]$ cette définition exprime un MDD à deux dimensions de 9 cellules et centré sur (0,0). Il va de « -1 » à « 1 » selon chaque dimension. Il y a donc 3x3 cellules :

(-1 ; -1)	(0 ; -1)	(1 ; -1)
(-1 ; 0)	(0 ; 0)	(1 ; 0)
(-1 ; 1)	(0 ; 1)	(1 ; 1)

Partiellement $[*:*, *:*, 0:*]$ cette définition exprime un MDD à trois dimensions dont les deux premières sont laissées ouvertes et la troisième commence à zéro et n'a pas de maximum.

Implicitement « 2 » cette définition exprime un MDD à deux dimensions. Elle est équivalente à l'expression partielle $[* : *, * : *]$.

Par exemple, la définition :

typedef marray `<octect, [-1:1, -1:1]> kernel3`

convient aux fenêtres de convolution 3x3. Alors que la définition suivante convient pour les images RVB :

typedef marray `<RGB, 2> RGBImg`

4.4.1.3 Les types d'ensembles

Enfin, vient le type d'ensembles. Il exprime simplement quel type de MDD la collection peut accueillir. La syntaxe est très simple :

typedef set `<£nom_du_type_de_mdd> £nom_du_type_de_l'ensemble`

Par exemple, une collection abritant des images RVB pourra être décrite de la sorte :

typedef set `<RGBImg> RGBSet`

4.4.1.4 Utiliser Rasdl

L'utilisation du langage Rasdl se fait via les API ou bien via le module du même nom. L'invocation du module peut prendre les paramètres suivants entre autres :

`−i` pour insérer des types dans la base de données.

`−r` pour aller lire les informations dans un fichier.

`−p` pour afficher les types de cellules, de MDD et d'ensembles qui sont déjà dans la base de données.

En général, l'insertion de nouveaux types se fait à partir d'un fichier :

```
rasdl -i -r fichier_de_types
```

Par défaut, le module `rasdl` se connecte à la base de données `RASBASE`. Il est prévu pour une utilisation locale et pas pour communiquer à travers le réseau. Ce module permet également de créer et supprimer les bases de données.

4.4.2 Rasml

Précisons d'entrée de jeu que de nombreuses similitudes existent entre le langage WCPS et le langage Rasml, notamment au niveau des fonctionnalités.

La syntaxe de Rasml est proche de celle de SQL. En toute généralité pour une requête `select` :

select `£liste_de_traitements`
from `£liste_de_collection`
`£[`**where** `£condition_boolenne]`

Rasdaman itère sur les collections fournies par le `from` en sélectionnant les MDD répondant à la condition fournie par le `where` et exécute les opérations fournies avec le `select`. Par exemple :

```
select tiff(mdd[100:150, 40:80] /2)
from RGBcol as mdd
where some_cell(mdd.red > 150)
```

extrait sous format TIFF, en divisant la valeur de chaque composante de chaque cellule par deux, tous les MDD contenus dans la collection nommée `RGBCol` dont la composante nommée « red » excède 150 pour au moins une cellule.

Quant aux requêtes d'insertion et de mise à jour :

```
insert into £collection
values £mdd

update £collection as £collection_iterateur
set £collection_iterateur[ £intervalle ] assign £mdd
```

Avant de passer aux opérations possibles et à la syntaxe de celles-ci, penchons nous d'abord sur les collections.

4.4.2.1 Création de collection

Une fois un type d'ensemble créé, on peut créer une collection basée sur lui :

```
create collection £nom_de_la_collection £type_de_l'ensemble
create collection RGBcol RGBSet
```

Les prochaines sections exposent les possibilités de Rasdaman en matière de manipulation de MDD. Ces possibilités seront revues dans un cadre plus géographique ultérieurement.

4.4.2.2 Les possibilités de Rasml : les filtres de données

La première opération possible est l'application de filtres d'entrée et sortie pour convertir les données. Ces filtres ont été exposés à la figure 4.4. Chaque format possède deux filtres, un d'entrée et un de sortie. L'application des filtres se fait via une fonction `x`() pour la sortie et `inv_x`() pour l'entrée. L'utilisation est la suivante :

```
select £x( £mdd )
from £collection as £mdd

insert into £collection values £inv_x( $1 )
```

$1 est une variable qui prendra le nom du fichier. Un exemple complet, utilisant le module `rasql`, sera présenté au point 4.4.2.13.

4.4.2.3 Les possibilités de Rasml : les opérations « spatiales »

Avant de passer aux opérations spatiales, enrichissons la syntaxe sur les intervalles vue précédemment (section 4.4.1.2). On parlera d'intervalle (monodimensionnel) lorsqu'on ne traite qu'une seule dimension. En revanche, on parlera d'intervalle multidimensionnel lorsqu'on en traitera plusieurs. Ainsi, un intervalle s'exprime comme [`a` : `b`] alors qu'un intervalle multidimensionnel s'exprimera (2D) :[`a` : `b`, `c` : `d`]. Il est possible d'extraire l'intervalle d'une dimension hors d'un intervalle multidimensionnel en faisant référence à sa dimension (la numérotation commence à zéro) :

```
[a:b, c:d][0]=[a:b]
[a:b, c:d][1]=[c:d]
```

Ensuite, il est possible d'extraire les bornes d'un intervalle monodimensionnel grâce aux commandes `.lo` et `.hi` :

```
[a:b].lo = a
[a:b, c:d][0].hi=b
```

Les opérations de *trimming* et *slicing* (fig. 2.6) sont très simples à mettre en place dans Rasdaman :

```
select £mdd[£sous_domaine_spatial]
from £collection as £mdd

select mdd[25:40, 5]
from ma_collection as mdd
```

L'exemple exécute une opération de *slicing* et de *trimming*. Le *trimming* se fait sur la première dimension où un sous-intervalle est choisi ($[25:40]$) alors que le *slicing* se fait sur la deuxième dimension. Le résultat sera donc un MDD 1D. Il est possible d'effectuer uniquement un *slicing* : $[*:*, 5]$.

Enfin, Rasdaman met à disposition quatre fonctions pour travailler avec le domaine spatial : `sdom`(), `shift`(), `extend`() et `scale`(). La première permet d'extraire le domaine spatial d'un MDD. Par exemple, si on voulait extraire la borne supérieure de la première composante des MDD de la collection `ma_collection`, il faudrait utiliser la requête suivante :

```
select sdom(mdd)[0].hi
from ma_collection as mdd
```

La deuxième fonction, `shift`(), permet d'effecteur une translation :

```
shitf(£mdd, £vecteur_de_translation)

update ma_collection as mdd
set mdd assign shift(mdd, [0,4])
```

Cette commande permet de mettre à jour tous les MDD de la collection `ma_collection` en effectuant une translation de quatre unités sur la deuxième dimension.

La fonction `extend`() permet d'étendre le domaine spatial d'un MDD. Le domaine fourni doit contenir le domaine du MDD et les pixels supplémentaires prennent la valeur nulle.

```
extend( $mdd, $intervalle_multidimensionel)

update ma_collection as mdd
set mdd assign extend(mdd, [0:5000, 0:6000])
```

Finalement, la fonction `scale`() permet d'effectuer des changements d'échelle. Elle peut s'utiliser de trois manières différentes. La première consiste à effectuer un changement d'échelle identique sur toutes les dimensions. La seconde manière consiste à produire un changement d'échelle asymétrique selon les dimensions. La dernière consiste à préciser la taille de l'image finale. Respectivement :

```
select scale( mdd, 0.5)
from ma_collection as mdd

select scale (mdd, [2, 0.75])
from ma_collection as mdd

select scale(mdd, [0:99, 0:99])
from ma_collection as mdd
```

4.4.2.4 Les possibilités de Rasml : les constantes et les changements de type

Tout comme en SQL, il est possible de glisser des constantes dans les requêtes. Avec Rasdaman, il faut préciser le type des constantes par l'ajout d'un suffixe comme en C/C++. La liste des suffixes est reprise à la figure 4.9. Par exemple, `123c` précise que le nombre `123` est un entier non signé codé sur 8 bits.

postfix char	type
c	char
o	octet
s	short
us	unsigned short
l	long
ul	unsigned long
f	float
d	double

FIGURE 4.9 – Liste des suffixes pour les constantes (Baumann 2009b)

Rasdaman permet évidemment de créer des constantes de types composés et des MDD. Une autre technique permettant de créer des MDD sera exposée plus loin (*cfr.* section 4.4.2.9). Pour les types composés, la syntaxe rappelle celle vue en Rasml :

```
struct { £constante_1,  £[£constante_n]*}
struct { 123c, 123c, 123c}
```

Et en ce qui concerne la définition des MDD, il est plus facile de l'illustrer. La matrice suivante

0	1	2
1	2	3
2	3	4
3	4	5
4	5	6

peut s'exprimer comme :

```
< [-1:1, -2:2] 0c, 1c, 2c, 3c, 4c ; 1c, 2c, 3c, 4c, 5c ;
    2c, 3c, 4c, 5c, 6c >
```

```
insert into ma_collection
values < [-1:1, -2:2] 0c, 1c, 2c, 3c, 4c ;
    1c, 2c, 3c, 4c, 5c ; 2c, 3c, 4c, 5c, 6c >
```

Comme toujours en informatique, la valeur « vrai » des booléens est exprimée par le nombre 1 et la valeur « faux » par le nombre 0.

En outre, il est possible d'effectuer des changements de types comme en C++. Cette opération est cependant limitée aux types primitifs :

```
(nouveau_type)valeur
```

L'exemple suivant permet de tronquer un réel pour en faire un entier :

```
(char)12.34d = 12
```

4.4.2.5 Les possibilités de Rasml : les opérations arithmétiques

Les opérations arithmétiques classiques $(+, -, *, /)$ sont bien évidemment présentes dans Rasdaman. Il est possible de les appliquer sur chaque cellule du MDD :

```
select $mdd $operateur $constante
from $collection as $mdd
```

ou de faire une opération cellule à cellule entre deux images :

```
select $mdd_1 $operateur $mdd_2
from $collection_1 as $mdd_1, $collection_2 as $mdd_2

select (mdd1 + mdd2) /2
from collection_1 as mdd1, collection_2 as mdd2
```

Notons qu'on peut rajouter l'opération `overlay` dans cette catégorie. Cette opération permet de superposer deux MDD de même dimension en plaçant le premier MDD au-dessus du second :

- ⋆ Si la valeur de la cellule du premier opérande est différente de la valeur nulle, c'est celle-là qui est choisie.
- ⋆ Si la valeur de la cellule du premier opérande est nulle, la valeur choisie est celle du second opérande.

4.4.2.6 Les possibilités de Rasml : les relations

La relation unaire `not` inverse un MDD booléen. Les relations binaires arithmétiques $(=, !=, <, >, <=, >=)$ et logiques (`and`, `or`, `xor`) s'exécutent entre cellules homologues des deux MDD. Toutes ces opérations renvoient un MDD booléen. Seules les opérations $=$ (testant l'égalité) et $!=$ (testant l'inégalité) sont définies pour les MDD de type composé.

La requête suivante renvoie une collection de MDD masquée par un seuil inférieur à `127`. Tous les pixels de valeur inférieure à `127` vaudront `0`.

```
select mdd * (mdd > 127)
from ma_collection as mdd
```

4.4.2.7 Les possibilités de Rasml : les fonctions usuelles

Rasdaman met à disposition les fonctions suivantes :

```
sqrt(),
abs(),
exp(), log(), ln(),
sin(), cos(), tan(),
arcsin(), arccos(), arctan().
```

Celles-ci représentent la plupart des fonctions qu'utilise WCPS. Il faut juste noter la disparition de la fonction puissance.

```
select £fonction( £mdd )
from £collection as £mdd
```

4.4.2.8 Les possibilités de Rasml : les condenseurs

Les condenseurs, comme le nom l'indique, permettent de condenser l'information. Rasdaman offre deux solutions pour mettre en place les condenseurs. La première consiste en une batterie de fonctions :

`add_cells`() renvoie la somme de toutes les cellules.

`avg_cells`() renvoie la moyenne de toutes les cellules.

`min_cells`() renvoie le minimum de toutes les cellules.

`max_cells`() renvoie le maximum de toutes les cellules.

`count_cells`() prend un MDD booléen et compte le nombre de cellules dont la valeur est « vraie ».

`some_cells`() prend un MDD booléen et renvoie « vrai » si au moins une cellule est « vraie ».

`all_cells`() prend un MDD booléen et renvoie « vrai » si toutes les cellules sont « vraies ».

Les cinq premiers s'utilisent avec un `select` alors que les deux derniers s'utilisent dans un `where` :

```
select avg_cells(mdd)
from ma_collection as mdd

select mdd
from ma_collection as mdd
where some_cells(mdd > 125)
```

Rasdaman offre également un mécanisme pour définir d'autres condenseurs :

```
condense £operateur
over £var in £intervalle_multidimensionel
£[where £condition]
using £expression_de_cellule
```

Le mécanisme est puissant mais difficile à appréhender. Rasdaman va itérer sur toutes les cellules du domaine défini par `£intervalle_multidimensionel` et, pour autant que la `£condition` soit respectée, calculer l'expression définie par `£expression_de_cellule` et finalement utiliser l'opérateur défini dans `£operateur` sur l'ensemble des cellules visitées. Un exemple sera plus clair : la fonction `some_cells` peut se réécrire comme :

```
select
    condense or
    over x in sdom(mdd)
    using mdd[x]
from ma_collection as mdd
```

Rasdaman exécute donc, pour chaque MDD appartenant à `ma_collection` l'opération `or` entre chaque pixel.

4.4.2.9 Les possibilités de Rasml : le constructeur

Comme nous l'avons déjà vu, il est possible de construire des MDD via le langage Rasml. La technique précédente (*cfr.* section 4.4.2.4) permettait de créer un MDD en spécifiant la valeur de chaque cellule. Le constructeur permet, quant à lui, de créer des MDD de manière algébrique. De plus, utilisé conjointement avec d'autres opérations, notamment les condenseurs, il permet d'effectuer des traitements plus complexes comme les fenêtres de convolutions. Des exemples de ces traitements complexes seront présentés lors du prototypage des traitements. La syntaxe du constructeur est la suivante :

```
marray £iterateur in £domaine_spatial
values £expression_de_cellule
```

La construction d'un MDD dont le domaine spatial est $[1:100, -50:200]$ dont toutes les cellules valent 1 et codé sur 8 bits non signés s'écrirait :

```
marray x in [1:100, -50:200]
values 1c
```

4.4.2.10 Faire référence à un MDD

Rasdaman offre un moyen pour faire référence à un MDD. Il assigne à chaque MDD (et à chaque collection) un OID (*Object IDentificator*) qu'il ne faut pas confondre avec les OID du SGBD-OR. L'OID du MDD peut s'utiliser de deux manières, soit pour obtenir l'OID (dans un `select`), soit pour sélectionner un MDD (dans un `where`)

```
Select OID(mdd)
from ma_collection as mdd

Select mdd
from ma_collection as mdd
where OID(mdd)=512
```

C'est par ce procédé qu'on peut lier des métadonnées aux MDD.

4.4.2.11 Faire référence à une composante

Si le MDD se base sur des cellules de type composé, il est possible de ne faire référence qu'à une seule composante via l'opérateur « . » :

```
£MDD.£nom_de_composante
£MDD.£numero_de_composante
```

La composante peut être référencée via son nom ou via sa position (la numérotation commence à 1 cette fois) :

```
select mdd.red
from RGBCol as mdd

select mdd.1
from RGBCol as mdd
```

4.4.2.12 Paramétrisation des tuiles

Les informations reprises dans cette section ne sont pas disponibles dans la documentation de Rasdaman. Elles viennent d'un article écrit par Mr. Baumann sur le sujet (Baumann 2010c). Par défaut, Rasdaman ne découpe pas les *rasters* introduits en tuiles. C'est donc la responsabilité de l'administrateur insérant les données de gérer :

- ⋆ S'il y a découpage ou non.
- ⋆ La stratégie de découpage.
- ⋆ L'indexation utilisée.
- ⋆ Le format sous lequel les données sont stockées.
- ⋆ La compression utilisée (non disponible dans la version *open source* et donc non discutée dans ce mémoire).

Tous ces choix sont offerts lors d'une requête d'insertion :

```
insert into £collection values £MDD £[£Directives]
```

Rasdaman offre cinq stratégies de découpage. La première découpe le *raster* de manière régulière (`regular`) : les tuiles sont les cellules d'une grille dont chaque ensemble de droites présente un espacement constant. En d'autres mots, il s'agit d'un découpage en *rasters* plus petits. Les tuiles ont toutes les mêmes dimensions à l'éventuelle exception de celles du bord. Cette stratégie de découpage est la plus performante si l'accès aux données ne suit pas un schéma particulier.

La deuxième découpe est proche de la précédente. Seule la contrainte de l'équidistance entre les droites définissant le maillage disparaît dans la stratégie `aligned`. Celle-ci est intéressante lorsqu'on ne veut pas de découpage selon certaines directions. Par exemple, dans le cas d'un MDD représentant une série temporelle d'images satellites, il sera peut-être intéressant de le découper selon l'axe temporel afin d'obtenir des tuiles qui représentent l'image entière pour une époque donnée.

La découpe `directional` est identique à la précédente. La différence entre les deux stratégies vient de la manière de spécifier les bornes. La méthode `aligned` se base sur la superficie des tuiles alors que la stratégie `directional` précise pour chaque dimension un ensemble de points par lesquels devront passer les axes de découpage.

La méthode `area of interest` permet de définir un ensemble de zones fortement sollicitées. Sur base de celles-ci, un découpage irrégulier est opéré de manière à stocker ces zones dans des tuiles individuelles. Si les zones d'intérêt se recouvrent, elles sont partitionnées en plusieurs tuiles plus petites.

La stratégie `area of interest` a comme problème qu'elle peut générer un grand nombre de tuiles même si le nombre de zones d'intérêts est limité. La dernière stratégie tend à résoudre ce problème en regroupant plusieurs zones d'intérêt pour autant qu'elles soient proches et pour autant que la nouvelle zone soit suffisamment grande. Cette technique s'appelle le `statistic` *tilling*.

Une fois la stratégie de découpage et ses paramètres spécifiés, il faut choisir la méthode d'indexation utilisée. Rasdaman en propose deux. La méthode par défaut se base sur un index `R+ tree` (`rpt_index` , Sellis *et al.* 1997). La seconde méthode, `rc_index`, valable seulement pour les trois premières méthodes, consiste en une simple table d'adressage listant pour chaque dimension les bornes de chaque tuile.

Finalement, Rasdaman permet de choisir le format sous lequel sont stockés les MDD : le format binaire par défaut qui ne demande pas d'encodage et de décodage, les formats PNG, TIFF et JPEG.

Par exemple, si on veut introduire une image de dimension 1024x600 sous quatre tuiles régulières au format PNG, la syntaxe serait :

```
insert into ma_collection values ($1)
tiling regular [ (0:511) , (0:299) ]
index rc_index
storage png
```

4.4.2.13 Utilisation de Rasml

Comme dit précédemment, Rasml s'utilise avec le module `rasql` ou via les API. Le module `rasql` utilise les paramètres principaux suivants :

`-q` permet de fournir la requête sous forme d'une chaîne de caractères.

`-f` permet de fournir un fichier auquel la variable `$1` fera référence (*cf.* 4.4.2.2).

`-out` permet de spécifier le format de sortie. Les valeurs possibles sont :

- `none` aucune sortie.
- `file` sortie d'un MDD sous forme de fichier. Suppose l'utilisation d'un filtre de sortie.
- `string` sortie en console sous forme d'une chaîne de caractères. Fonctionne seulement pour certaines requêtes (`sdom`, `OID`) et pour les MDD 1D de type `char`.
- `hex` sortie en console sous forme hexadécimale.

`--user` renseigne le nom d'utilisateur. Défaut : `rasguest`. Il permet l'extraction mais pas l'insertion.

`--passwd` renseigne le mot de passe. Défaut : `rasguest`.

D'autres options permettent de fournir, entre autres, l'adresse de l'hôte ainsi que le numéro de port écouté par le médiateur. Par défaut, ces valeurs sont respectivement `localhost` et `7001`. Une requête classique d'extraction ou d'insertion se fait de la sorte :

```
rasql -q 'select mdd from ma_collection as mdd'
rasql -q 'insert into ma_collection values inv_tiff($i)'
    -f input/mon_image.tiff --user rasadmin --passwd rasadmin
```

4.4.2.14 Retour sur les métadonnées

Comme on peut le voir et en conformité avec ce qui a été dit précédemment, Rasdaman n'a pas la capacité pour gérer lui-même les métadonnées. Ceci sera encore revisité lors de l'exploration du modèle logico-physique de Rasdaman.

4.5 La gestion interne des données

Afin de pouvoir offrir la souplesse dont il a été question au niveau de l'approche conceptuelle, la gestion interne des données de Rasdaman est complexe. De plus, elle n'est pas documentée ; les informations qui vont suivre ont été obtenues par *retro engineering* en comparant l'évolution de la base de données aux changements effectués via Rasdaman (insertion et suppression d'information). Cette étape a été réalisée simultanément à un premier prototypage des données. Néanmoins, elle est placée ici pour des raisons de cohérence.

Cette section va s'articuler, dans un premier temps, sur les mêmes notions que le modèle conceptuel de Rasdaman et va suivre le même ordre que la définition des données en Rasdl (4.4.1). Dans un second temps, on s'intéressera à la manière dont sont stockées les données.

4.5.1 Le stockage des types

Le stockage des types est repris à la figure 4.10.

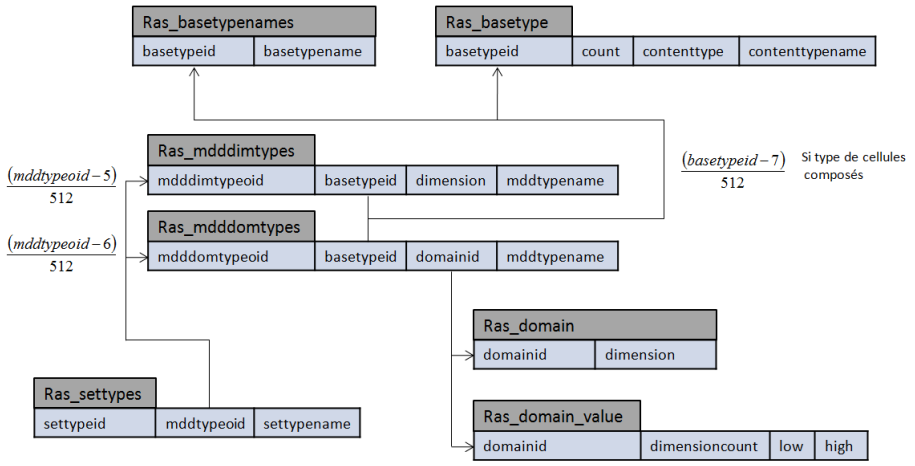


FIGURE 4.10 – Schéma des tables entrant dans la gestion des types

4.5.1.1 Les types de cellules

Deux tables servent à la description des types composés :

```
ras_basetypenames(basetypeid, basetypename)
ras_basetype(basetypeid, count, contenttype, contenttypename)
```

La table `ras_basetypenames` associe à chaque type composé un identifiant. La seconde table, `ras_basetype`, décrit chaque type composé. Elle se base sur le même identifiant (`basetypeid`) que la première table et pour chaque composante (`count`), elle décrit son type primitif (`contenttype`) et son nom (`contenttypename`). Le type primitif est exposé via un code. Par exemple, `1040` correspond au type `char`. Ce code est interne à Rasdaman et n'est détaillé nulle

« identifiant »	nom	Contenttype : $512id + 16$
0	`unsigned long`	16
1	`unsigned short`	528
2	`char`	1040
3	`boolean`	1552
4	`long`	2064
5	`short`	2576
6	`octet`	3088
7	`double`	3600
8	`float`	4112
9	`complex`	4624
10	`complexd`	5136

TABLE 4.1 – Relation linéaire entre les types primitifs et les `contenttypes` correspondants

part. Néanmoins, on peut observer une relation linéaire entre les `contenttypes`. Celle-ci est reprise au tableau 4.1.

4.5.1.2 Les types de MDD

Un type de MDD définit la dimensionnalité, éventuellement des restrictions sur l'extension selon certaines dimensions ainsi que le type des cellules. S'il n'y a pas de restrictions sur le domaine spatial, c'est la table suivante qui stocke les informations sur le type de MDD :

```
ras_mdddimtypes(mdddimtypeoid, basetypeid, dimension,
    mddtypename)
```

Elle définit un identifiant (`mdddimtypeoid`) pour chaque type de MDD, le type de cellules (`basetypeid`), la dimensionnalité (`dimension`) et bien sûr le nom du type de MDD (`mddtypename`).

En revanche, s'il y a des restrictions, ce sont les tables suivantes qui s'en chargent :

```
ras_mdddomtypes(mdddomtypeoid, basetypeid, domainid,
    mddtypename)
ras_domain(domainid, dimension)
ras_domain_value(domainid, dimensioncount, low, high)
```

Les tables `ras_domain` et `ras_domain_value` gèrent tout ce qui concerne les domaines spatiaux ; chaque fois qu'un domaine spatial est créé, il atterrit dans ces tables. Elles seront encore utilisées ultérieurement. La première, `ras_domain`, attribue pour chaque domaine un identifiant et sa dimensionnalité. La seconde, `ras_domain_value`, reprend, pour chaque dimension la borne inférieure et supérieure[1]. Dans le cas présent, le but de ces deux tables est de stocker les restrictions sur le domaine spatial.

1. Les bornes inférieures et supérieures sont décrites par des entiers. Il n'est donc pas possible d'utiliser des coordonnées continues.

La table `ras_mdddomtypes`, quant à elle, gère les autres informations. Les attributs `mddtypename` et `basetypeid` ont les mêmes fonctions que précédemment. L'attribut `mdddomtypeoid` sert d'identifant et l'attribut `domainid` permet de faire le lien avec les tables `ras_domain` et `ras_domain_value`.

Le `basetypeid` utilisé dans les tables `ras_mdddomtypes` et `ras_mdddimtypess` est en lien avec les deux tables concernant le type de cellules : `ras_basetypenames` et `ras_basetype`. Lorsqu'il s'agit d'un type primitif, l'identifiant fait référence au `contenttype` de la table `ras_basetype` repris dans le tableau 4.1. En revanche, s'il s'agit d'un type composé, le `basetypeid` des tables `ras_mdddimtypes` et `ras_mdddomtypes` est alors calculé à partir du `basetypeid` des tables `ras_basetypenames` et `ras_basetype` selon la formule suivante :

$$ras\_mdddimtypes.basetypeid = 512\, ras\_basetypenames.basetypeid + 7$$
$$ras\_mdddomtypes.basetypeid = 512\, ras\_basetypenames.basetypeid + 7$$

4.5.1.3 Les types d'ensemble

Pour rappel, le type d'ensemble ne fait qu'attribuer un nom à un ensemble pouvant contenir un type de MDD donné. Il est décrit par une seule table :

```
ras_settypes(settypeid, mddtypeoid, settypename)
```

La table n'offre pas vraiment de surprises. Elle associe un identifiant (`settypeid`) à chaque ensemble (`settypename`) et référence le type de MDD qu'elle peut contenir (`mddtypeoid`).

L'attribut `mddtypeoid` est dérivé à partir des attributs `mdddomtypeoid` de la table `ras_mdddomtypes` et `mdddimtypeoid` de la table `ras_mdddimtypes` :

$$ras\_settypes.mddtypeoid = 512\, ras\_mdddimtypes.mddtypeoid + 5$$
$$ras\_settypes.mddtypeoid = 512\, ras\_mdddomtypes.mdddomtypeoid + 6$$

4.5.2 Le stockage des informations

Le stockage des informations est repris à la figure 4.11.

4.5.2.1 Les collections

Deux tables servent à décrire les collections :

```
ras_mddcollnames(mddcollid, settypeid, mddcollname)
ras_mddcollection(mddid, mddcollid)
```

Comme on peut s'en douter, `ras_mddcollnames` contient les informations des collections alors que `ras_mddcollection` associe les MDD à leur collection.

La table `ras_mddcollnames` reprend pour chaque collection son nom (`mddcollname`), un identifiant (`mddcollid`) et le type d'ensemble qui caractérise la collection (`settypeid`). Ce dernier est identique à celui trouvé dans la table `ras_settypes`.

Les identifiants repris dans la table `ras_mddcollection` ne souffrent pas non plus d'une transformation linéaire. L'attribut `mddcollid` est donc le même que son homonyme de la table `ras_mddcollnames`. Et l'attribut `mddid` est identique à celui de la table `ras_mddobjects` (*cf.* le point suivant) du même nom.

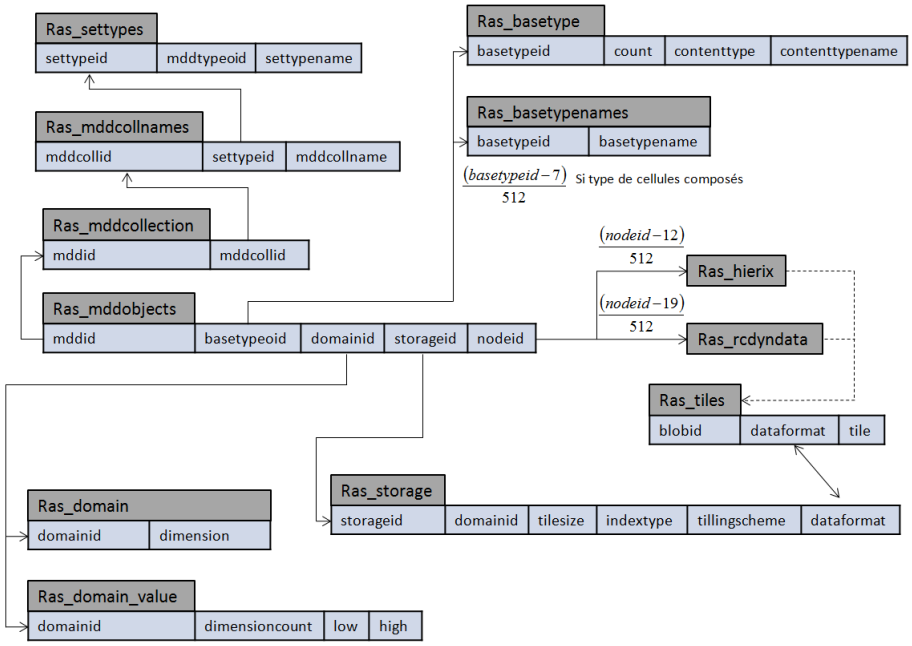


FIGURE 4.11 – Schéma des tables entrant dans le stockage des informations

4.5.2.2 Les MDD

Le stockage des MDD est complexe et dépend de l'indexation utilisée. Dans tous les cas, chaque MDD a une entrée dans les tables :

```
ras_mddobjects(mddid, basetypeoid, domainid, storageid, nodeid)
ras_storage(storageid, domainid, tilesize, indextype,
    tillingscheme, dataformat)
```

La table `ras_mddobjects` contient des informations sur le MDD. Parmi celles-ci, on retrouve un identifiant (`mddid`), le type des cellules (`basetypeoid`), le domaine spatial (`domainid`), l'OID fourni par Rasml (`storageid`) et un identifiant permettant de retrouver les informations des tuiles (`nodeid`).

Le type des cellules est dérivé à partir des tables `ras_basetypenames` et `ras_basetype` comme décrit plus haut (4.5.1.1 et 4.5.1.2). L'attribut `domainid` fait référence à l'attribut homonyme des tables `ras_domain` et `ras_domain_value` qui décrit le domaine spatial du MDD. L'attribut `storageid` est calculé à partir de l'attribut `mddid` sur base de la formule $storageid = 512\,mddid + 11$. Notons cependant que l'OID fourni par Rasml est légèrement différent : $OID = 512\,mddid + 1$. Enfin, l'attribut `nodeid` permet d'identifier un *tuple* dans les tables responsables des indexations spatiales. Nous en rediscuterons après avoir décrit la table `ras_storage`.

Cette dernière contient les informations en rapport avec le stockage des MDD. L'identifiant `storageid` est identique à l'attribut `mddid` de la table `ras_mddobjets`. L'attribut `domainid` permet de décrire le découpage en tuiles via les tables `ras_domain` et `ras_domain_value`. L'attribut `tilesize` spécifie

la taille maximale d'une tuile (en octet). L'attribut `indextype` spécifie l'indexation utilisée. L'attribut `tillingscheme` spécifie quel découpage en tuiles a été utilisé (*cfr.* section 4.4.2.12). Finalement, `dataformat` spécifie sous quel format sont stockés les MDD.

Comme dit précédemment, en fonction de l'indexation spatiale l'attribut `nodeid` fera référence à différentes tables. S'il est de la forme $nodeid = 512\,k+12$, alors il s'agit d'une indexation de type `R+ tree` (ce qui est aussi mentionné dans la table `ras_storage`) et les informations sont reprises dans la table `ras_hierix`. Cette table contient un élément `dyndata` qui est un OID de PostgreSQL. C'est cet élément binaire qui contient l'index. En revanche, si l'attribut `nodeid` est de la forme $nodeid = 512\,k + 19$, alors l'indexation est de type `rc` et le fichier binaire d'indexation est repris dans la table `ras_rcindexdyn` dans l'attribut `dyndata`.

Et en ce qui concerne le stockage des tuiles, c'est la responsabilité de la table

```
ras_tiles(blobid, dataformat, tile)
```

Celle-ci contient un identifiant (`blobid`), le format des données (`dataformat`) qui est le même que dans la table `ras_storage` et un OID de PostgreSQL contenant la tuile (`tile`).

Les OID de PostgreSQL sont des attributs particuliers. Il s'agit d'un entier servant de clé externe vers une métatable contenant des BLOB (Postgresql 2008).

4.6 PetaScope : le serveur WCPS

La notoriété de Rasdaman ne vient pas seulement de ses capacités de serveur de données *rasters* mais également de ses capacités comme serveur web implémentant les standards de l'OGC. Rasdaman est d'ailleurs revendiqué comme le seul serveur WCPS (Baumann et Aiord 2010) existant à nos jours.

Au niveau de la structure, le serveur WCPS, PetaScope, est en fait une extension de Rasdaman et ne constitue pas un composant de base. C'est pourquoi, il n'a pas été évoqué plus tôt. PetaScope se présente sous la forme d'une *servlet* Java se plaçant au dessus de Rasdaman (le serveur *raster* sur la figure 4.12). Le but de PetaScope est double. D'une part, il doit traduire les requêtes WCPS et WCS en requêtes compréhensibles par Rasdaman (donc en Rasml). D'autre part, il gère également un ensemble de métadonnées dont certaines sont à caractères géographiques et viennent combler les manques de Rasdaman en la matière.

On peut cependant regretter le manque de documentation spécifique à PetaScope.

4.6.1 Le modèle logico-physique de PetaScope

Pour son implémentation, PetaScope utilise 15 tables supplémentaires. Les six premières tables sont statiques ; elles ne doivent pas être modifiées. Elles contiennent des informations sur les capacités de PetaScope. Les tables restantes servent à décrire les *rasters* gérés par PetaScope. La figure 4.13 reprend le schéma conceptuel de PetaScope. Celui-ci était présent parmi les fichiers de

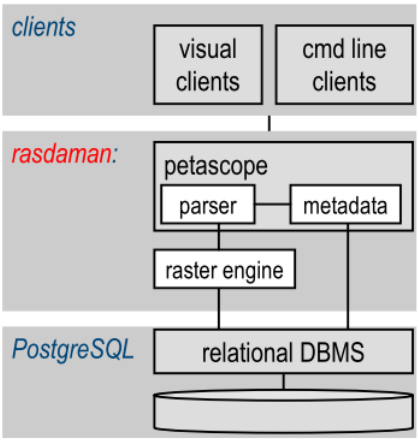


FIGURE 4.12 – Place de PetaScope dans l'architecture de Rasdaman (Baumann 2010a)

PetaScope. Assez ironiquement, le modèle logico-physique de PetaScope est la seule documentation existante sur PetaScope alors que celui de Rasdaman n'est détaillé nulle part.

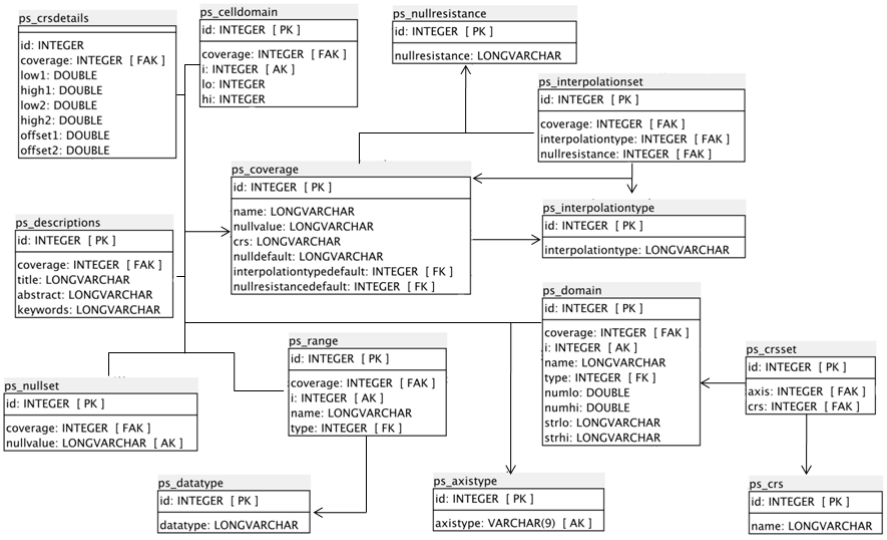


FIGURE 4.13 – Schéma conceptuel de PetaScope

4.6.1.1 Les tables statiques

Les six tables statiques sont les suivantes :

```
ps_axistype(id, axistype)
ps_datatype(id, datatype)
```

```
ps_interpolationtype(id, interpolationtype)
ps_nullresistance(id, nullresistance)
ps_crs(id, name)
ps_format(id, name, mimetype)
```

La table `ps_axistype` décrit ce que représente chaque axe : l'axe « x », l'axe « y », l'axe temporel, l'axe des altitudes,... La table `ps_datatype` définit les types primitifs que gère PetaScope. Ceux-ci sont évidemment les mêmes que ceux de Rasdaman (fig. 4.7b). La table `ps_interpolationtype` décrit les interpolations possibles : aucune, plus proche voisin, linéaire, cubique et quadratique. Allant de pair avec celle-ci, la table `ps_nullresistance` décrit les types de *null resistance* qui correspondent à la manière dont sont gérées les valeurs nulles lors de l'interpolation (OGC 2008). La table `ps_crs` liste les systèmes de coordonnées que PetaScope utilise. Cette table est très décevante car elle ne contient que deux *tuples*, le CRS correspondant à l'image et celui correspondant à WGS84. De plus, PetaScope dispose de la table pour être en conformité avec le standard mais ne l'utilise pas et gère les *rasters* uniquement en coordonnées discrètes. Finalement, `ps_format` fait l'inventaire des formats dans lesquels PetaScope peut délivrer ses *rasters* : `csv`, `png`, `jpeg`, `tiff`, ou sous forme d'un flux d'octets.

4.6.1.2 Les tables de métadonnées

Les tables dynamiques décrivent les *coverages* situés dans la base de données. Pour insérer un *raster* dans la base de données, il faut d'abord l'introduire via Rasdaman dans une collection dont il est le seul élément. Ensuite, il faut insérer les métadonnées dans les tables de PetaScope. La première table,

```
ps_coverage(id, name, crs, nulldefault,
    interpolationtypedefault, nullresistancedefault)
```

reprend l'identifiant du *raster*. Le nom du *raster* doit être identique à la collection le contenant. C'est ainsi que PetaScope fait le lien avec l'information *raster* déjà introduite via Rasdaman. C'est cet attribut qui servira d'identifiant lors des requêtes WC(P)S. L'attribut `crs` décrit évidemment le système de référence. Notons qu'il ne s'agit pas d'une clé externe vers l'identifiant de la table `ps_crs` mais d'un élément textuel. La gestion du système de coordonnées est prise en charge par une autre table. Enfin, les attributs `interpolationtypedefault` et `nullresistancedefault` sont respectivement des clés externes vers les tables `ps_interpolationtype` et `ps_nullresistance`.

La table suivante décrit l'extension spatiale du *raster* dans le système image :

```
ps_celldomain(id, coverage, i, lo, hi)
```

Chaque *tuple* a un identifiant et se rapporte à un *coverage*. L'attribut `coverage` est une clé externe vers l'identifiant du *raster* dans la table `ps_coverage`. L'attribut `i` définit la dimension décrite par le *tuple*. Quant aux deux derniers, ils permettent de stocker les bornes de chaque axe. Notons que cette table est totalement redondante puisque Rasdaman garde les mêmes informations sur le domaine spatial de chaque *raster*.

Allant de pair avec la table précédente, la table

```
ps_domain(id, coverage, i, name, type, numlo, numhi, strlo,
    strhi)
```

gère le domaine spatio-temporel des *rasters*. Les trois premiers attributs (`id`, `coverage` et `i`) ont exactement la même fonction que dans la table précédente. L'attribut `name` donne le nom de l'axe alors que l'attribut `type` est une clé externe vers la table `ps_axistype`. S'il ne s'agit pas de l'axe temporel, alors les attributs `numlo` et `numhi` reprennent respectivement la borne inférieure et supérieure, du domaine selon cet axe et les attributs `strlo` et `strhi` sont laissés vides. En revanche, s'il s'agit d'un axe temporel, ce sont les attributs `numlo` et `numhi` qui sont vides et les attributs `strlo` et `strhi` qui contiennent les informations. Dans l'état actuel des choses, PetaScope ne se sert pas des quatre derniers attributs puisqu'il ne gère aucun aspect des coordonnées géographiques.

Les deux tables suivantes sont en lien avec les systèmes de coordonnées. Même si PetaScope ne les gère pas pour le moment, la structure des tables est déjà là :

```
ps_crsset(id, axis, crs)
ps_crsdetails(id, coverage, low1, high1, low2, high2,
    offset1, offset2)
```

La première table permet d'associer à chaque axe (`axis` est une clé étrangère vers la table `ps_domain`, même si le nom est trompeur) un système de référence (`crs` est une clé étrangère vers la table `ps_crs`). Quant à la deuxième table, elle permet de fournir des informations pour les transformations de coordonnées en fournissant le cadre capable du *raster* dans le système WGS84.

En vis-à-vis des tables décrivant le domaine spatial, on trouve une table décrivant le *range* :

```
ps_range(id, coverage, i, name, type)
```

Les deux premiers attributs sont toujours les mêmes : un identifiant du tuple et une clé externe permettant d'identifier le *raster*. L'attribut `i` identifie la composante du type de cellule. S'il s'agit d'un type composé, plusieurs *tuples* feront donc référence au même *raster*. L'attribut `name` donne le nom de la composante. Enfin, l'attribut `type` est une clé externe vers la table `ps_datatype`. Une fois encore, cette table est redondante puisque toutes ces informations sont déjà présentes dans Rasdaman.

La table

```
ps_interpolationset(id, coverage, interpolationtype,
    nullresistance)
```

décrit pour chaque *raster* quelle interpolation est possible. Les deux premiers attributs ont les significations habituelles. Les attributs `interpolationtype` et `nullresistance` sont respectivement des clés externes vers les tables `ps_interpolationtype` et `ps_nullresistance`. L'intérêt de cette table est qu'elle permet de définir d'autres interpolations que celle par défaut reprise dans la table `ps_coverage`. Néanmoins, l'interpolation par défaut doit faire partie de la table.

L'avant dernière table est :

```
ps_nullset(id, coverage, nullvalue)
```

Elle permet de préciser l'ensemble des valeurs représentant une valeur nulle dans le *raster*. PetaScope ne fait pas la distinction entre valeur nulle et absence de valeur. Tout comme la table précédente, la valeur nulle par défaut doit être présente dans cette table et dans la table `ps_coverage`.

Finalement, la table

`ps_description(id, coverage, title, abstract, keywords)`

reprend les informations permettant de décrire le *raster*.

4.6.1.3 Les limitations de PetaScope

PetaScope présente deux limites importantes. La première est le fait qu'il n'existe aucun mécanisme automatique pour charger la base de données. Il faut donc remplir les tables les unes après les autres par requêtes SQL. On pourrait, cependant, grandement faciliter le processus via l'écriture de scripts.

La seconde limitation, plus contraignante pour le domaine de la géomatique, est celle que nous avons déjà relevée : PetaScope ne gère pas les systèmes de références et donc toutes les requêtes s'en servant. Afin de respecter le standard et de pouvoir facilement implémenter le support des systèmes de références, les tables sont déjà prêtes mais pour l'instant elles ne servent pas. Pour cause, Mr. Baumann explique dans le groupe de discussion de Rasdaman que l'OGC va bientôt retravailler sur les systèmes de références :

I am working on the OGC specification for CRS support ; this actually will not only affect WCS, but I am in contact with SWE and WMS folks as well so as to come up with an OGC-wide solution. The issue is that, (not only) for multi-dimensional data, there are requeriments for ad-hoc CRSs and parametric CRSs. This transcends currents, mostly EPSG-based, CRSs. Further, we need to establish and OGC registry of CRS definitions. A draft spec. will be available in September, and we will implements this in parrallel (ie. before adoption) as rasdaman is a WCS reference implementation. So stay tuned...

Si les systèmes de références ne sont pas pris en charge pour le moment, cela ne devrait tarder.

Malheureusement, je n'ai jamais réussi à faire fonctionner PetaScope. Après plusieurs vérifications complètes de toutes les bibliothèques pré-requises et l'installation de différentes versions de PetaScope, j'ai abandonné la piste pour ne pas compromettre le reste du mémoire. Soulignons cependant que, si PetaScope apporte un meilleur support des métadonnées et l'aspect web à Rasdaman, il ne règle pas complètement le problème de celles-ci.

4.7 Confrontation partielle aux critères de validation

Nous allons commencer par prendre un peu de temps pour résumer ce qui a été dit dans cette présentation et puis nous le confronterons aux critères de validation.

Rasdaman est un serveur de données permettant de gérer des MDD. Ceux-ci sont des structures inspirées des tableaux informatiques qui permettent de gérer des *rasters* de dimensionnalité et taille *a priori* infinis. De plus, les MDD peuvent contenir des cellules dont le type est composé de plusieurs types primitifs.

Rasdaman est un *middleware* et le stockage des informations est délégué à un SGBD, dans notre cas, PostgreSQL. Le rôle de Rasdaman est la gestion des MDD uniquement. Il ne prend donc pas en charge les métadonnées. Ainsi, Rasdaman se rapproche du paradigme des architectures hybrides, même si toutes les données sont stockées dans la même base de données. Cela se reflète dans le langage de requêtes inspiré de SQL qui semble très riche mais qui est limité aux MDD. Cela se reflète également dans les filtres de données qui ne gèrent pas vraiment les métadonnées. En particulier, l'aspect géographique des informations est perdu.

Finalement, la gestion interne des *rasters* est complexe et Rasdaman n'offre pas d'outil d'aide à la conception ni de modélisateur de traitements.

Il est maintenant possible d'effectuer une comparaison partielle avec les critères de validation. En ce qui concerne les fonctionnalités, Rasdaman répond à presque tous les critères obligatoires. Le point du catalogue de données sera rediscuté par la suite. Concernant les données, Rasdaman s'en sort *a priori* très bien celles qu'il peut gérer. En revanche, il ne gère pas du tout les métadonnées. Ce constat est le plus important désavantage de Rasdaman jusqu'à présent et va avoir un impact sur les traitements et les requêtes; les requêtes sur les les métadonnées ne sont évidemment pas possibles, les projections cartographiques non plus. À l'inverse, les requêtes se limitant à la matrice de valeurs semblent possibles.

Nous allons maintenant passer au prototype pour tester réellement les capacités de Rasdaman.

Chapitre 5

Le prototype

Dans cette partie, nous allons concevoir et implémenter le prototype. Il s'agit donc de tester les capacités réelles de Rasdaman d'un point de vue qualitatif et quantitatif. Ce chapitre est divisé en quatres sections.

La première décrira l'environnement dans lequel le prototype a été développé ainsi que l'installation de Rasdaman et la manière dont seront évaluées les performances.

La seconde section s'intéressera aux données à insérer dans le serveur. Les données utilisées seront décrites et confrontées à celles mentionnées dans la liste des critères de validation (*cfr.* section 3.3). Elles seront alors chargées dans le serveur et les performances mesurées. Une dernière partie comparera les objectifs à ce qui a été réalisé.

La troisième section est consacrée aux traitements et aux requêtes. La liste de ceux à soumettre au prototype est dressée en rapport avec les critères de validation. Ces traitements et requêtes sont alors passés en revue et leur performance mesurée. La section se termine par une brève conclusion.

Enfin, la dernière section fait le point sur ce qui a pu être appris grâce au prototype.

5.1 Environnement de travail

Le prototype est réalisé sur un poste fixe dans l'environnement suivant :

- ⋆ Ordinateur AMD Athlon 64, 2 GHz et 1024 MB de mémoire vive.
- ⋆ Système d'exploitation Linux Ubuntu Jaunty -9.04.
- ⋆ Rasdaman version 8.0.
- ⋆ PostgreSQL 8.3.12.

Une nouvelle version de Rasdaman est disponible depuis le 28 juillet 2011 mais n'apporte pas de modifications majeures.

5.1.1 Installation

L'installation de PostgreSQL est bien documentée dans son manuel. Celle de Rasdaman est plus complexe. Deux manuels lui sont dédiés. Le premier est le manuel d'installaion à proprement parler. Le second est le manuel d'intégration

avec PostgreSQL. Les informations sont parfois contradictoires entre les deux manuels et le celui d'intégration n'est pas à jour.

Il est conseillé de créer un compte d'utilisateur au nom de `rasdaman`. Une fois PostgreSQL en place, il faut installer tous les paquets requis par Rasdaman. La liste de ceux-ci est présentée dans le manuel d'installation (qui précise qu'il vaut mieux aller voir la liste des paquets sur le site de Rasdaman). Une fois installés, il faut télécharger l'archive contenant Rasdaman. La procédure d'installation est décrite dans le manuel d'installation. Il s'agit de décompresser l'archive, compiler le programme et l'installer. Une fois l'opération réussie, reste à configurer quelques variables d'environnement.

Il faut alors passer à l'intégration avec PostgreSQL. Là aussi, quelques variables d'environnement ont besoin d'être configurées. Elles sont décrites dans le manuel d'intégration. Il faut alors configurer PostgreSQL pour qu'il accepte les connexions TPC/IP. Le manuel d'intégration n'est pas à jour sur ce point. Il faut se reporter à la documentation de PostgreSQL. Les configurations se font dans les fichiers `postgresql.conf` et `pg_hba.conf`. Le premier fichier permet d'autoriser les connections TCP/IP et le second permet de restreindre les utilisateurs accédant à la base de données.

À partir de là, on peut revenir au manuel d'installation pour initialiser la base de données. Le script `create_db.sh` crée une base de données du nom de `RASBASE` et y charge les tables et les types par défaut.

Rasdaman est alors prêt pour fonctionner. Il faut d'abord lancer le médiateur (`rasmgr`) et puis démarrer les serveurs. Le script `start_rasdaman.sh` est fourni à cet effet.

5.1.2 Mesures de performances

Au vu des possibilités de Rasdaman, les mesures de performances seront réalisées avec la commande `time` de linux qui permet d'obtenir des informations sur le temps que prend une autre commande pour s'exécuter. La précision de la commande est de l'ordre du centième de second donc largement suffisante dans le cadre de notre application.

Les requêtes seront exécutées via la commande `rasql`. C'est donc elle qui fait l'objet de la mesure. Celle-ci est légèrement biaisée du fait qu'elle ne porte pas seulement sur le temps de la requête mais sur tous les éléments suivants :

- ⋆ La connexion entre le client et le médiateur.
- ⋆ La redirection depuis le médiateur vers un `rasserver` et la connexion à ce dernier.
- ⋆ L'envoie des données au `rasserver`.
- ⋆ Le traitement de la requête à proprement parlé.
- ⋆ Le renvoi du résultat.

Cette remarque avait pour vocation de résoudre une ambiguité de langage. Le biais en lui même ne pose pas de problème : toutes ces étapes doivent s'effectuer quelle que soit la requête. Les mesurer ne porte donc pas préjudice. De plus, les mesures s'effectuant en local, elles ne sont pas entâchées d'un bruit lié au réseau. Dès lors, quand on parlera de la mesure du temps des requêtes, nous ferons référence à toutes ces étapes.

5.2 Prototypage des données

Nous entamons maintenant la conception du prototype sur l'aspect des données. Cette section est divisée en quatre sous-sections. La première explique le jeu de données qui servira au prototype. La seconde section présente le modèle conceptuel des données. La troisième est relative au chargement des données. Finalement, la dernière partie fait le point sur le prototypage.

5.2.1 Choix des données

Le choix des données a été motivé par les critères concernant les données, les traitements et les requêtes. La localisation des images n'était donc pas un facteur de choix.

- ⋆ Deux images 1024x600 Landsat TM codées sur 8 bits et voisines. Celles-ci servent à tester :
  - Les capacités de stockage des images codées sur 8 bits : images binaires, nominales et quantitatives discrètes.
  - Les capacités de mosaïquage.
  - Des requêtes et traitements divers.
- ⋆ Quatre images 1024x600 SPOT formant une image multi-spectrale, toujours codées sur 8 bits. Ceci permet de tester :
  - Les structures multi-spectrales.
  - Les compositions colorées.
  - Des requêtes et traitements divers.
- ⋆ Cinq images Quickbird, quatre canaux multi-spectraux (256x150) et une image panchromatique (1024x600), codées sur 10 bits. Ceci permet de :
  - Tester les capacités de stockage des images codées sur 10 bits.
  - Tester des requêtes et traitements simples.
  - Discuter l'organisation des données à résolutions différentes.
- ⋆ Un modèle numérique de surface, codé en entiers sur 16 bits, exprimant l'altitude en décimètres et faisant 1024x600. Ceci permet de :
  - Tester les capacités de stockage vis-à-vis des *rasters* codés en réels.
  - Tester des traitements sur les *rasters* codés en réels.
  - Discuter du rééchantillonnage.

Le modèle de surface fourni exprimait l'altitude en décimètres et était donc codé en entier sur 16 bits. L'altitude a été réexprimée en mètres et l'image codée en réels sur 32 bits.

Les données sont sous format GeoTIFF. Ces données permettent de tester l'ensemble des critères de validation. En effet, les images binaires peuvent être dérivées de celles-ci et les images nominales sont codées sur 8 bits. Les images diachroniques sont gérées de la même manière que les images multi-spectrales par Rasdaman. Tous les traitements restants (c'est-à-dire tous, excepté les projections cartographiques) peuvent être traités avec le présent jeu de données.

5.2.2 Le modèle conceptuel du prototype

Cette section décrit les choix effectués pour organiser les données en collections au sein du prototype. Pour chaque collection, on précise le type d'ensembles, le type de MDD et le type de cellules. Si le cas utilisé n'est pas par

défaut dans la base de données, il est détaillé ici. Les traitements prévus spécifiquement pour certaines collection sont également mentionnés.

5.2.2.1 La collection `landsatcol`

La collection `landsatcol` doit abriter les deux images Landsat. Comme il s'agit d'images de type simple codées sur 8 bits, il existe déjà dans la base de données un type d'ensemble la gérant : `GreySet` basé sur le type de MDD `GreyImage` qui est un MDD 2D basé sur le type de cellule primitif `char` (entiers non signés codés sur 8 bits).

Cette collection est expressément prévue pour tester le mosaïquage.

5.2.2.2 La collection `mns`

La collection `mns` servira à stocker le modèle numérique de surface. Depuis la conversion en réels, la collection peut se baser sur le type d'ensemble `FloatSet` basé sur le type de MDD `FloatImage` lui même basé sur le type primitif de cellules `float`.

Cette collection permettra de discuter de l'interpolation via le changement d'échelle, de créer des masques et des reclassifications. Il servira églament pour un calcul de pentes qui sera l'opération la plus compliquée réalisée par le prototype.

5.2.2.3 La collection `spotcol`

La collection `spotcol` est une collection qui sert pour stocker les images SPOT individuellement. Elle sera donc constituée de quatre MDD reposant sur un type primitif de cellules. Le type d'ensemble sera le même que pour les images Landsat : `GreySet`.

L'intérêt de cette collection résidera notamment dans la comparaison avec les autres collections basées sur les images SPOT. Elle servira à tester la création de compositions colorées et d'histogrammes sur deux canaux.

5.2.2.4 La collection `spot3`

La collection `spot3` est notre première collection utilisant un type composé. Elle va contenir les trois premières images SPOT. Son intérêt est uniquement de montrer une insertion d'images RGB dans la base de données. En effet, une composition colorée créée sur les trois premières bandes avec le logiciel Idrisi sera insérée dans la base de données. Un type d'ensemble est déjà présent dans Rasdaman pour gérer ce cas, il s'agit du type `RGBSet` basé sur le type de MDD `RGBImage`, lui-même basé sur un triplet de type de cellules primitif `char`.

5.2.2.5 La collection `spot4`

La collection `spot4` va cette fois contenir les quatre canaux dans un seul MDD. C'est le premier type composé qu'il faut définir :

```
struct UB4bands {char first, second, third, fourth; };
typedef marray <UB4bands, 2> UB4BImage;
typedef set <UB4BImage> UB4BSet;
```

Tout d'abord, il faut définir le type des cellules : `UB4bands` pour *Unsigned Bytes over four bands*, il s'agit des quatre bandes qui sont de type entier non signé. Ensuite, il faut définir le type de MDD : `UB4BImage` qui est donc MDD 2D composé de cellules de types `UB4bands`. Finalement, il faut définir le type d'ensemble : `UB4BSet` qui est un ensemble reprenant des MDD de type `UB4BImage`.

Les traitements effectués sur cette collection seront les mêmes que sur la collection `spotcol` afin de pouvoir comparer les syntaxes et performances.

5.2.2.6 La collection `spot3D`

La collection `spot3D`, à ne pas confondre avec la collection `spot3`, sera également constituée d'un seul MDD. Cette fois, il s'agira d'un MDD 3D. Les deux premières dimensions représenteront le domaine spatial et la troisième représentera le domaine spectral. Les cellules sont donc du type primitif entier non signé. La définition d'une telle structure est la suivante :

```
typedef marray <char, 3> pseudo3DImage;
typedef set <pseudo3DImage> pseudo3DSet;
```

Cette fois, il n'est pas nécessaire de définir le type des cellules. Quant aux traitements, il s'agira à nouveau des mêmes traitements que `spotcol` et `spot4` pour pouvoir comparer les syntaxes et les performances.

5.2.2.7 La collection `quickcol`

La collection `quickcol` va stocker les différentes images Quickbird sous forme de cinq MDD 2D de type primitif de cellules. Son rôle est de vérifier la capacité de Rasdaman à stocker les images codées sur 10 bits. La structure pour stocker ces images est la suivante :

```
typedef marray <unsigned short, 2> USImage;
typedef set <USImage> USSet;
```

Notons que pour Rasdaman et pour GDAL les images Quickbird correspondent à des entiers non signés codés sur 16 bits.

`quickcol` est la seule collection reprenant les images Quickbird car la bande panchromatique n'a pas la même résolution spatiale que les autres. Il est donc impossible, sans l'aide d'artefact, de créer un MDD reprenant toutes les bandes sous forme d'un type composé de cellules. Il est également impossible de créer une collection qui contiendrait les bandes multi-spectrales dans un seul MDD et la bande panchromatique dans un second MDD car ceux-ci n'auraient alors pas le même type de cellules.

Les nouveaux types définis peuvent être placés dans un fichier ASCII et chargés dans la base de données via le module `rasdl` comme expliqué à la section 4.4.1.4. Cette étape est instantanée.

5.2.3 Chargement des données

Une fois insérés les types d'ensemble susmentionnés, le chargement des données à proprement parlé peut commencer. Celui-ci s'effectue en deux parties. D'abord il faut construire les collections qui viennent d'être décrites et puis il faut insérer les données dans la base de données.

5.2.3.1 La création des collections

Pour rappel, la création des collections s'effectue suivant la syntaxe :

create collection `£nom_de_collection £type_d'ensemble`

Via le module `rasql`, la syntaxe complète est, par exemple :

```
rasql -q 'create collection landsatcol GreySet'
    --user rasadmin --passwd rasadmin
```

Il ne faut pas oublier de spécifier l'utilisateur ayant les droits d'insertions dans la base de données et son mot de passe (`rasadmin`). La création des collections est une opération instantanée. L'ensemble des requêtes donne :

```
create collection landsatcol GreySet
create collection mns FloatSet
create collection spotcol GreySet
create collection spot3 RGBSet
create collection spot4 UB4BSet
create collection spot3D pseudo3DSet
create collection quickcol USSet
```

5.2.3.2 L'insertion des données : les cas simples

Les collections se basant sur des MDD primitifs et la collection `spot3` forment les cas simples. Leur insertion n'a pas posé de souci. Les requêtes d'insertions sont les suivantes pour :

⋆ Les images Landsat :

```
rasql -q 'insert into landsatcol values inv_tiff($1)'
    -f input/Donnees/Istanbul-LANDSAT-2000-07-02/p180r031.tif
        --user rasadmin --passwd rasadmin
rasql -q 'insert into landsatcol values inv_tiff($1)'
    -f input/Donnees/Istanbul-LANDSAT-2000-07-02/p180r032.tif
        --user rasadmin --passwd rasadmin
```

⋆ Le modèle numérique de terrain :

```
rasql -q 'insert into mns values inv_tiff($1)'
    -f input/Donnees/MNS_RW/msn_truereal32.tif
        --user rasadmin --passwd rasadmin
```

⋆ Les images Spot individuelles :

```
rasql -q 'insert into spotcol values inv_tiff($1)'
    -f input/Donnees/Istanbul-SPOT-XS10m-2009-10-02/xs1.tif
       --user rasadmin --passwd rasadmin
rasql -q 'insert into spotcol values inv_tiff($1)'
    -f input/Donnees/Istanbul-SPOT-XS10m-2009-10-02/xs2.tif
        --user rasadmin --passwd rasadmin
rasql -q 'insert into spotcol values inv_tiff($1)'
    -f input/Donnees/Istanbul-SPOT-XS10m-2009-10-02/xs3.tif
        --user rasadmin --passwd rasadmin
rasql -q 'insert into spotcol values inv_tiff($1)'
    -f input/Donnees/Istanbul-SPOT-XS10m-2009-10-02/xs4.tif
        --user rasadmin --passwd rasadmin
```

⋆ Les images Quickbird :

```
rasql -q 'insert into quickcol values inv_tiff($1)'
    -f input/Donnees/QuickBird/qb-pan.tif
        --user rasadmin --passwd rasadmin
rasql -q 'insert into quickcol values inv_tiff($1)'
    -f input/Donnees/QuickBird/qb-xs1.tif
        --user rasadmin --passwd rasadmin
rasql -q 'insert into quickcol values inv_tiff($1)'
    -f input/Donnees/QuickBird/qb-xs2.tif
        --user rasadmin --passwd rasadmin
rasql -q 'insert into quickcol values inv_tiff($1)'
     -f input/Donnees/QuickBird/qb-xs3.tif
         --user rasadmin --passwd rasadmin
rasql -q 'insert into quickcol values inv_tiff($1)'
     -f input/Donnees/QuickBird/qb-xs4.tif
        --user rasadmin --passwd rasadmin
```

⋆ Les images Spot pour la collection `spot3` (après composition colorée sur les trois premières bandes) :

```
rasql -q 'insert into spot3 values inv_tiff($1)'
    -f input/Donnees/Istanbul-SPOT-XS10m-2009-10-02/
        spot_rgb.tif
            --user rasadmin --passwd rasadmin
```

L'insertion d'une image individuelle ne cause aucun problème. Rasdaman compare le type de la collection dans laquelle on souhaite insérer les données et celles qu'il reçoit via le paramètre `−f`. S'ils sont compatibles, Rasdaman les insère dans la base de données. Sinon, une exception de type

```
MDD and collection types are incompatible
```

est lancée. Même les images de type RVB peuvent être insérées sans souci. Le tableau 5.1 reprend pour chacune des insertions la taille physique des images et le temps d'insertion. Comme on peut le voir, le chargement est très rapide. La première des images landsat a cependant pris un temps important pour être insérée par rapport aux autres.

À regret, on peut noter que l'insertion/extraction d'une image dans la base de données n'est pas neutre. En particulier, les informations à caractères géographiques sont perdues :

Les métadonnées d'une image SPOT avant insertion :

```
Driver: GTiff/GeoTIFF
Files: xs1.tif
Size is 1024, 600
Coordinate System is:
PROJCS["WGS 84 / UTM zone 35N",
 GEOGCS["WGS 84",
  DATUM["WGS_1984",
   SPHEROID["WGS 84",6378137,298.2572235629972,
    AUTHORITY["EPSG","7030"]],
   AUTHORITY["EPSG","6326"]],
```

Image insérée	Taille des images	Temps d'insertion [s]
Landsat 1	603.2 ko	1.08
Landsat 2	603.2 ko	0.22
Modèle numérique de surface	2.3 Mo	0.65
Spot 1	603.2 ko	0.46
Spot 2	603.2 ko	0.24
Spot 3	603.2 ko	0.2
Spot 4	603.2 ko	0.24
La composition colorée des images Spot	1.8 Mo	0.55
Quickbird 1	76.8 ko	0.14
Quickbird 2	76.8 ko	0.13
Quickbird 3	76.8 ko	0.12
Quickbird 4	76.8 ko	0.11
Quickbird panchromatique	1.2 Mo	0.25

TABLE 5.1 – quelques mesures sur les premières insertions

```
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]],
 PROJECTION["Transverse_Mercator"],
 PARAMETER["latitude_of_origin",0],
 PARAMETER["central_meridian",27],
 PARAMETER["scale_factor",0.9996],
 PARAMETER["false_easting",500000],
 PARAMETER["false_northing",0],
 UNIT["metre",1,
  AUTHORITY["EPSG","9001"]],
 AUTHORITY["EPSG","32635"]]
Origin = (656960.000000000000000,4546070.000000000000000)
Pixel Size = (10.000000000000000,-10.000000000000000)
Metadata:
AREA_OR_POINT=Area
TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
 @(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
  $Date: 2004/09/15 18:42:01EDT $
TIFFTAG_XRESOLUTION=1
TIFFTAG_YRESOLUTION=1
TIFFTAG_RESOLUTIONUNIT=1(unitless)
Image Structure Metadata: INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 656960.000, 4546070.000)
 ( 28d52'3.73"E, 41d 3'2.66"N)
Lower Left ( 656960.000, 4540070.000)
 ( 28d51'58.24"E, 40d59'48.19"N)
Upper Right ( 667200.000, 4546070.000)
 ( 28d59'22.14"E, 41d 2'55.32"N)
```

```
Lower Right ( 667200.000, 4540070.000)
 ( 28d59'16.29"E, 40d59'40.86"N)
Center ( 662080.000, 4543070.000)
 ( 28d55'40.10"E, 41d 1'21.82"N)
Band 1 Block=1024x8 Type=Byte, ColorInterp=Gray
```

Les métadonnées de la même image après exportation :

```
Driver: GTiff/GeoTIFF
Files: output/xs1-out.tif
Size is 1024, 600
Coordinate System is ''
Metadata:
TIFFTAG_DOCUMENTNAME=exported from rasdaman database
TIFFTAG_SOFTWARE=rasdaman
TIFFTAG_ARTIST=rasdaman
Image Structure Metadata:
INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  600.0)
Upper Right ( 1024.0,    0.0)
Lower Right ( 1024.0,  600.0)
Center      (  512.0,  300.0)
Band 1 Block=1024x8 Type=Byte, ColorInterp=Gray
```

Les deux collections restantes (`spot4` et `spot3D`) sont plus complexes à insérer, comme nous allons le voir.

5.2.3.3 L'insertion des données : la collection `spot4`

Pour rappel, le but de cette collection est de construire un MDD avec un type de cellules composé. La première et principale difficulté de l'opération vient du fait qu'aucun mécanisme n'est précisé dans le langage Rasml pour gérer cette situation. La seconde difficulté est que Rasdaman ne permet pas, du moins pour le moment (Baumann 2009b), de fournir plusieurs fichiers lors d'une requête avec le module `rasql`. Cet obstacle pourrait être facilement contourné en insérant un MDD vide (*i.e.* rempli de zéros) ou partiellement vide (une seule composante est insérée et le reste est rempli par la valeur nulle) et puis, en mettant à jour chaque composante séparément. Néanmoins, se pose toujours le problème de la syntaxe à employer.

Examinons d'abord le cas d'une insertion partielle. La première possibilité consiste à insérer une seule composante et à laisser Rasdaman remplir de valeurs nulles le reste du MDD. La syntaxe d'une telle requête serait :

```
insert into spot4.0 values inv_tiff($1)
```

Comme un message d'erreur le signale, l'insertion n'est pas prévue pour fonctionner de la sorte. Toujours en essayant une insertion partielle, on peut tenter de spécifier la structure à Rasdaman de la sorte :

```
insert into spot4 values struct{inv_tiff($1),0c,0c,0c}
insert into spot4 values marray x in [0:1023, 0:599]
    values struct{inv_tiff($1),0c,0c,0c}
```

Cependant, ces requêtes ne fonctionnent pas mieux car la syntaxe `struct{...}` est prévue pour les constantes.

On peut alors passer au cas d'une insertion vide suivie de mises à jour composante par composante :

```
insert into spot4 values struct{0c, 0c, 0c,0c}
```

Une fois encore, plusieurs possibilités sont envisageables :

```
update spot4 as s set s.0 assign inv_tiff($1)
update spot4 as s set s.0 assign s.0 + inv_tiff($1)
```

Celles-ci ne fonctionnent pas car « `set s.0` » n'est pas valide syntaxiquement.

```
update spot4 as s set s assign s.0 + inv_tiff($1)
```

ne fonctionne pas non plus car les types sont incompatibles : on essaie de mettre à jour des cellules de type composé avec un élément primitif. L'opération additionne la première composante avec l'image TIFF fournie, ce qui donne un MDD primitif. Ce MDD ne peut servir à la mise à jour du MDD précédent puisque les types ne se correspondent pas. Si on essaie l'instruction `struct`, on retombe sur le même problème que précédemment .

Les formules suivantes fonctionnent :

```
update spot4 as s set s assign {0c, 0c, 0c, 0c}
update spot4 as s set s assign s + marray x in [0:1023, 0:599]
    values struct{0c, 0c, 0c, 0c}
```

Alors que les mêmes formules utilisant les fichiers d'insertion ne fonctionnent pas car l'instruction `struct` définit des constantes :

```
update spot4 as s set s assign {inv_tiff($1), 0c, 0c, 0c}
update spot4 as s set s assign s + marray x in [0:1023, 0:599]
    values struct{inv_tiff($1),0c,0c,0c}
```

Ainsi, comme nous pouvons le voir, ni via l'opérateur de sélection des composantes, ni via l'instruction `struct`, il n'est possible de mettre à jour ou d'insérer un MDD à structure composée.

Heureusement, il reste une possibilité : créer un fichier multi-bandes mais de plus de trois bandes cette fois. Le format GeoTIFF supporte ce genre de compositions. En revanche, Rasdaman ne le supporte pas ; s'il gère le TIFF, il n'est pas capable de gérer plus de trois bandes. L'insertion échoue donc car il ne comprend pas le fichier fourni.

Le problème a été soumis au groupe de discussion de Rasdaman mais l'auteur du logiciel, Mr. Baumann, n'a su apporter aucune solution jusqu'à présent. Il semble que pour le moment seules deux possibilités existent pour insérer des MDD de type composés : l'insertion par trois bandes (*cfr.* la collection `spot3`) ou bien l'insertion explicite. Cette dernière serait cependant un peu lourde à mettre en place puisque pour des images 1024x600 sur quatre canaux, cela requièrerait l'entrée de plus de deux millions de chiffres... Il est toujours possible de programmer un script permettant d'extraire les valeurs de l'image et de préparer la requête d'insertion explicite mais c'est regrettable que Rasdaman ne gère pas ça lui-même.

La limitation que nous avons vue ici (l'impossibilité de créer dynamiquement des MDD composés à partir de MDD simples) signifie que la composition colorée

n'est pas possible. Un autre impact, plus immédiat, est que la collection `spot4` n'a pas su être implémentée. Afin de pouvoir illustrer les traitements sur les structures composées, la collection `spot3` prendra la place de la collection `spot4`.

5.2.3.4 L'insertion des données : la collection `spot3D`

Nous passons maintenant à une autre collection dont le but était d'abriter des MDD 3D dont une dimension était dédiée aux différentes bandes spectrales. Rasdaman ne permet toujours pas d'utiliser plusieurs fichiers lors d'une insertion. Il faut donc insérer une structure vide et puis mettre à jour petit à petit :

```
insert into spot3D values marray x in [0:1023, 0:599, 0:3]
    values 0c
update spot3D as s set s[0:1023,0:599,0] assign inv_tiff($1)
update spot3D as s set s[0:1023,0:599,1] assign inv_tiff($1)
update spot3D as s set s[0:1023,0:599,2] assign inv_tiff($1)
update spot3D as s set s[0:1023,0:599,3] assign inv_tiff($1)
```

La différence avec le cas de `spot4`, c'est qu'ici il faut sélectionner une partie du domaine et non pas une composante. Si la requête est juste syntaxiquement, elle ne produit cependant pas l'effet escompté lors de l'extraction. La requête

```
select tiff(mdd[*:*, *:*, 0]) from spot3D as mdd
```

produit une image déformée (fig. 5.1). Seulement la moitié supérieure de l'image est conservée, réduite de deux sur l'axe des colonnes et répétées deux fois.

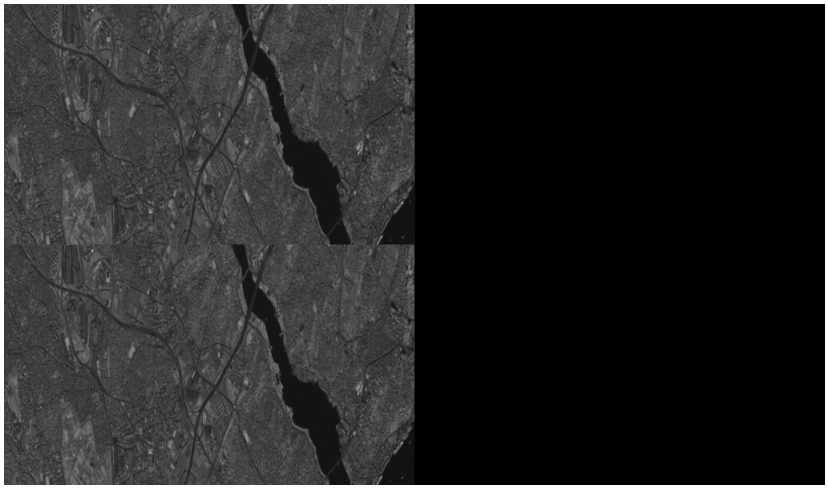


FIGURE 5.1 – Déformations lors de l'insertion d'une image par mise à jour directe

La requête suivante, qui comme la précédente aurait dû fournir le bon résultat, en fournit un tout aussi inatendu et de surcroit différent (après réinitialisation) :

```
update spot3D as s set s[*:*, *:*, 0] assign s[*:*, *:*, 0]
    + inv_tiff($1)
```

Les déformations observées sont identiques dans la moitié gauche de l'image alors que de nouvelles déformations apparaissent dans la moitié droite de l'image (fig. 5.2).

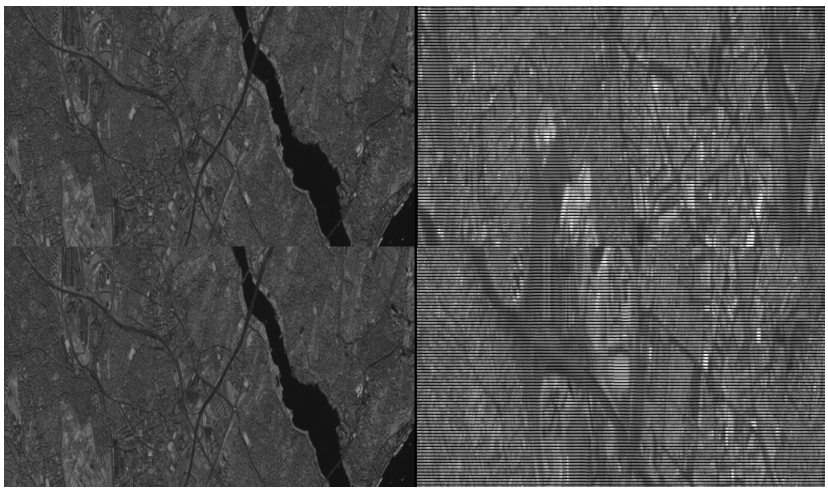


FIGURE 5.2 – Déformations lors de l'insertion d'une image par mise à jour indirecte

Le problème ne semble pas venir du filtre car une requête telle que :

```
update landsatcol as s set s assign inv_tiff($1)
```

fonctionne correctement quand le MDD est 2D. Le problème doit donc venir de l'itération sur le domaine spatial. En théorie, vérifier cette hypothèse aurait pû être simple. Il suffisait de réaliser l'insertion (ou la mise à jour) depuis la collection `spotcol`. En pratique, ce n'est pas possible et ceci nous permet de mettre le doigt sur une autre faiblesse du langage qui était restée cachée : l'impossibilité d'imbriquer des requêtes d'insertion ou de mises à jour avec des requêtes d'extraction. Ainsi des requêtes du type :

```
update spot3D as s
set s[*:*, *:*, 0]
assign (select mdd
    from spotcol as mdd
    where oid(mdd)=15361)
```

ou

```
update spot3D as s
set s[*:*, *:*, 0]
assign s[*:*, *:*, 0]  +
    (select mdd
    from spotcol as mdd
    where oid(mdd)=15361)
```

échoueront car on ne peut imbriquer des requêtes. Il est donc difficile de certifier que le dysfonctionnement vient de l'itérateur. Une fois encore, ce problème a

été soumis au groupe de discussion de Rasdaman mais aucune solution ne s'est encore dégagée.

Pour l'heure, nous devons conclure que la collection `spot3D` ne sera pas utilisable. Malheureusement, cette fois aucune autre ne pourra reprendre ses traitements comme ça avait été le cas pour `spot4` (repris par `spot3`).

S'il est dommage de ne pas pouvoir bénéficier de toute la flexibilité de Rasdaman en matière de gestion des *rasters*, rappelons nous tout de même que l'insertion sous forme de MDD individuels a réussi pour toutes les données.

5.2.4 Le bilan du prototypage des données

Nous pouvons désormais procéder à un petit bilan sur le prototypage des données. Hors de quatre groupes de données, sept collections devaient être créées afin de tester les capacités de Rasdaman pour stocker des données diverses dans des structures variées, d'une part. Et d'autre part, de tester un ensemble de traitements spécifiques à ces collections.

Cinq de ces collections ont pu être implémentées, ce qui représente la totalité du jeu de données et permet la réalisation de presque tous les traitements. En effet, la tentative d'implémentation de la collection `spot4` nous a appris que Rasdaman ne permettait pas la création dynamique de MDD non constant ayant un type de cellule composé. Ceci raie donc de la liste des traitements la composition colorée qui consistait précisément à créer un tel MDD sur base de trois MDD individuels. Mis à part celui-là, les autres traitements semblent envisageables.

On a également appris que pour autant qu'il soit possible, le chargement des données était rapide mais pas sans perte puisque les métadonnées des *rasters* disparaissaient, comme l'indiquait la partie sur la présentation de Rasdaman (chapitre 4). De plus, nous avons eu le temps de nous familiariser avec les langages de Rasdaman. Nous allons pouvoir passer au prototypage des traitements.

5.3 Le prototypage des traitements et requêtes

Nous poursuivons en nous intéressant aux traitements et requêtes. Le but de cette section est de mettre en évidence quels traitements sont réalisables et dans ce cas, quelle en est la syntaxe et quelles en sont les performances. Les traitements et requêtes suivantes ont été retenues :

- ⋆ Extractions des données
- ⋆ Changements de type de cellules
- ⋆ Extractions de statistiques
- ⋆ Opérations binaires
- ⋆ Masquages et reclassifications
- ⋆ Mosaïquages
- ⋆ Changements d'échelle et interpolations
- ⋆ Améliorations de contraste
- ⋆ Fenêtres de convolution
- ⋆ Requêtes sur les MDD

Ces traitements et requêtes sont en lien avec les critères de validations. Deux traitements ont disparu : les reprojections et la composition colorée. Puisque Rasdaman ne gère pas les systèmes de références, il ne sait pas effectuer les

changements de projection. La composition colorée, nous l'avons vu, est impossible car il n'y a pas moyen de créer dynamiquement un MDD composé à partir de plusieurs MDD primitifs.

Le domaine des requêtes a également dû être réduit (*cfr.* section 4.7 sur la confrontation partielle aux critères de validation). La seule métadonnée disponible directement depuis Rasdaman est le domaine spatial du MDD (dans le sens de Rasdaman). Il est néanmoins possible de calculer des statistiques, comme nous allons le voir. Par contre, toutes les autres métadonnées ne sont pas accessibles. Les requêtes se basant sur elles ne sont dès lors pas possibles.

5.3.1 Quelques requêtes importantes

Comme nous venons de le rappeler, la seule métadonnée disponible est le domaine spatial du MDD. Celui-ci peut être obtenu, pour la collection `spotcol`, par exemple, grâce à la requête :

```
select sdom(mdd) from spotcol as mdd
```

Exécutée avec le module `Rasql` :

```
rasql -q 'select sdom(mdd) from spotcol as mdd' --out string
```

Rasdaman dispose d'une seconde information importante : l'OID des MDD. Ces OID permettent de sélectionner seulement certains MDD lors d'une requête. On les obtient via la requête :

```
select oid(mdd) from spotcol as mdd
```

Enfin, Rasdaman offre une requête atypique permettant de lister les collections :

```
select col from RAS_COLLECTIONNAMES as col
```

`RAS_COLLECTIONNAMES` (à bien écrire en majuscule) n'est pas une vraie collection. Il s'agit d'une requête particulière de Rasdaman.

Il n'y a pas moyen d'obtenir d'autres informations sur les collections ou sur les MDD, pas même le type de ceux-ci ou le nombre de MDD au sein d'une collection. Cette information ne peut être connue que via une requête telle l'extraction du domaine spatial ou de l'OID des MDD. De même, l'utilisateur doit connaître l'ordre d'insertion des MDD pour savoir quel MDD possède quel OID. Avec la capacité d'afficher tous les types de cellules, de MDD et d'ensembles créés (via le module `rasdl`), la dernière requête est ce que Rasdaman offre de plus proche d'un catalogue de données. En l'absence de PetaScope, le critère se rapportant au catalogue de données n'est donc pas rempli.

5.3.2 Extraction des données

Dans cette sous-section, nous allons voir comment extraire les informations de la base de données. Des requêtes similaires parsèment ce mémoire mais nous allons ici en mesurer les performances. Nous allons commencer par l'extraction d'un MDD complet puis par une partie de MDD et enfin par les valeurs de cellules.

L'extraction complète d'un MDD de la collection `landsatcol` au format TIFF, s'effectue de la manière suivante :

```
select tiff(mdd) from landsatcol as mdd
```

Exécutée avec le module `rasql` :

```
rasql -q 'select tiff(mdd) from landsatcol as mdd' --out file
```

Pour rappel, nous avons vu à la section 5.2.3.2 que l'insertion et l'extraction ne sont pas neutres vis-à-vis des métadonnées.

L'extraction partielle d'un MDD se formule très simplement en spécifiant le domaine spatial du sous MDD. Par exemple, si on veut le sous MDD reprenant les 500 premières lignes et colonnes du MDD :

```
select tiff(mdd[0:500, 0:500]) from landsatcol as mdd
```

Il est également possible d'extraire la valeur d'une cellule. Par exemple, si on veut obtenir la valeur de la cellule aux coordonnées (25, 25) :

```
select mdd[25,25] from landsatcol as mdd
```

Attention, il ne s'agit plus d'extraire un MDD mais seulement une valeur. Il faut donc omettre le convertisseur et changer les paramètres de sorties de `rasql` :

```
rasql -q 'select mdd[25,25] from landsatcol as mdd'
    --out string
```

Les extractions sont des opérations rapides, comme le montrent les statistiques qui vont suivre. Celles-ci sont basées sur un échantillon de 100 requêtes :

Temps [s]	extraction	extraction partielle	une cellule
Minimum	0.2	0.13	0.06
Maximum	0.36	0.19	0.08
Moyenne	0.21	0.13	0.08
Écart type	0.02	0.01	< 0.01

5.3.3 Les changements de type de cellules

Ce troisième traitement cherche simplement à effectuer des changements de type de cellules. Ces changements s'effectueront sur la collection `quickcol`. La syntaxe est très simple :

```
select tiff((char)mdd) from quickcol as mdd
```

L'opération convertit un MDD dont les cellules sont codées sur 16 bits en un MDD dont les cellules sont codées sur 8 bits et puis extrait ce MDD sous forme d'image TIFF. Cette conversion s'accompagne évidemment d'une perte : les valeurs excédant 255 sont tronquées à ce seuil. Les statistiques suivantes ont été obtenues sur un échantillon de 100 requêtes pour l'image panchromatique (1024x600).

Temps [s]	Changement de types
Minimum	0.15
Maximum	0.67
Moyenne	0.32
Écart type	0.06

5.3.4 Quelques calculs de statistique

Ce quatrième groupe de traitement vise à extraire quelques statistiques des images. Ces traitements s'effectueront sur la collection `landsatcol` et seront validés à l'aide du module `gdalinfo` de GDAL.

5.3.4.1 Les informations de contrôles

GDAL fournit les informations suivantes :

	Landsat 1	Landsat 2
Minimum	1	1
Maximum	255	255
Moyenne	64.19	64.14
Écart type	19.71	19.71

5.3.4.2 Minimum, maximum et moyenne

Rasdaman fournit des fonctions permettant d'extraire le minimum, le maximum et la moyenne d'un MDD : `min_cells`(), `max_cells`() et `avg_cells`(). Les requêtes sont donc

```
select min_cells(mdd) from landsatcol as mdd
select max_cells(mdd) from landsatcol as mdd
select avg_cells(mdd) from landsatcol as mdd
```

La requête, via le module `Rasql` donne (pour le minimum) :

```
rasql -q 'select min_cells(mdd) from landsatcol as mdd'
    --out string
```

Ces requêtes fournissent les bonnes réponses. Les statistiques suivantes ont été calculées à partir d'un échantillon de 100 requêtes :

Temps [s]	`min_cells`()	`max_cells`()	`avg_cells`()
Minimum	0.13	0.14	0.15
Maximum	0.27	0.17	0.27
Moyenne	0.15	0.15	0.16
Écart type	< 0.01	< 0.01	< 0.01

Comme on peut le voir, ces opérations sont très rapides. Exécutées sur un MDD composé, l'opération renvoie un élément de type composé reprenant la valeur minimale de chaque composante. Par exemple, les trois images SPOT qui constituent la collection `spot3` admettent pour minimum : 14, 53 et 73. La fonction `min_cells`() appliquée à la collection `spot3` renvoie :

```
Query result collection has 1 element(s):
    Result element 1 : {14, 53, 73}
```

5.3.4.3 L'écart type

Pour l'écart type, la situation se complique car Rasdaman ne fournit pas de fonctions permettant de le calculer.. Il faut dès lors implémenter cette fonctionnalité. Pour se faire, on dispose de deux formules :

$$\sqrt{\frac{\sum_{i=1}^{n}(x_i-\overline{x})^2}{n-1}} = \sqrt{\frac{\sum_{i=1}^{n}x_i^2-\left(\sum_{i=1}^{n}x_i\right)^2/n}{n-1}}$$

où x représente la valeur d'une cellule, n correspond au nombre de cellules et $\overline{x}$ équivaut à la moyenne des valeur des cellules.

La première requiert la connaissance *a priori* de la moyenne et est moins précise numériquement (l'application de l'exposant avant la sommation). Nous étudierons d'abord le second cas.

Nous allons calculer chaque terme individuellement et puis recréerons la formule complète. Le premier terme du numérateur est $\sum x_i^2$, la somme des carrés des valeurs des cellules. Il peut être calculé de la sorte :

```
select
    condense +
    over x in sdom(mdd)
    using (unsigned long) mdd[x] * (unsigned long) mdd[x]
from landsatcol as mdd
```

On demande donc d'effectuer la somme du carré de la valeur de la cellule pour chaque cellule du domaine. Il est important de ne pas oublier de changer le type des cellules. La collection `landsatcol` est codée sur 8 bits ce qui est totalement insuffisant pour stocker le résultat. L'évaluation de la requête prend cette fois un certain temps (de l'ordre de 7.5 secondes pour deux MDD) et fournit au final :

```
Query result collection has 2 element(s):
    Result element 1 : 2770121627
    Result element 2 : 2766452307
```

Le second terme : $\frac{\left(\sum x_i\right)^2}{n}$ peut s'obtenir de la sorte :

```
select ((double) add_cells(mdd) * (double) add_cells(mdd))
    /(double)((sdom(mdd)[0].hi-sdom(mdd)[0].lo + 1)
    *(sdom(mdd)[1].hi-sdom(mdd)[1].lo) + 1)
from landsatcol as mdd
```

La fonction `add_cells`() permet de calculer la somme des cellules pour le numérateur. Le gros de l'instruction consiste à calculer le nombre de cellules. `sdom(mdd)` permet d'extraire le domaine spatial du MDD. `sdom(mdd)[0]` et `sdom(mdd)[1]` permettent d'obtenir l'intervalle monodimensionnel pour la première et deuxième dimension. Enfin, `sdom(mdd)[0].lo` et `sdom(mdd)[0].hi` permettent d'obtenir les bornes inférieures et supérieures. On calcule donc, pour chaque dimension, le nombre de cellules.

La réponse est plus rapide (de l'ordre 0.2 secondes) :

```
Query result collection has 2 element(s):
    Result element 1 : 2.53804e+09
    Result element 2 : 2.53448e+09
```

Si on contrôle ce résultat, on s'aperçoit que de grosses imprécisions se sont propagées. Par exemple, pour le premier MDD : $\frac{\sqrt{2770121627-2.53804\cdot10^9}}{1024\cdot600-1} = 19.43 \neq 19.71$. Ces erreurs sont dues au changement de type. En effet, si on extrait seulement la somme des cellules :

```
select add_cells(mdd) from landsatcol as mdd
```

on obtient 39436667. Donc

$$\sqrt{\frac{\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2 / n}{n-1}} = \sqrt{\frac{2770121627 - (39436667)^2 / (1024 \cdot 600)}{1024 \cdot 600 - 1}} = 19.71$$

ce qui est la bonne valeur. Si on ne change pas le type des cellules en réel codé sur 64 bits, les nombres sont trop grands et conduisent à une réponse fausse. Il parait logique qu'exécuter la requête globale engendrera des erreurs encore plus colossales. Néanmoins, il m'a été impossible de le vérifier car la requête consomme trop de ressources. Elle a tourné pendant plus de dix minutes sans venir au bout de ses opérations. La dite requête :

```
select sqrt(
    condense +
    over x in sdom(mdd)
    using (unsigned long)mdd[x] *(unsigned long)mdd[x]
    -((double) add_cells(mdd) * (double) add_cells(mdd))
    /(double)((sdom(mdd)[0].hi-sdom(mdd)[0].lo +1)
    *(sdom(mdd)[1].hi-sdom(mdd)[1].lo +1))
from landsatcol as mdd
```

En ce qui concerne la première formule de l'écart type ($\sqrt{\frac{\sum_{i=1}^{n}(x_i-\overline{x})^2}{n-1}}$), elle permet de trouver la bonne valeur d'écart type si la moyenne et le nombre de pixels sont connus :

```
select
    sqrt(
        condense +
        over x in sdom(mdd)
        using (((double) mdd[x] - 64.19d)
             *((double) mdd[x] - 64.19d)))
/614399d)
from landsatcol as mdd
```

Cette requête prend environ 10 secondes. Remarquons la lettre suffixe aux constantes pour indiquer qu'il s'agit d'un nombre réel codé sur 64 bits. Par contre, le développement du nombre de pixels (`count_cells`()) amène des erreurs (19.74 à la place de 19.71) et le calcul de la moyenne au sein de la requête produit le même souci que dans l'autre formule : trop de ressources consommées. Il semble donc qu'au cours de l'optimisation de la requête, Rasdaman ne calcule pas une seule fois la moyenne mais doive la recalculer pour chacune des 614400 itérations de la boucle.

En ce qui concerne l'écart type, on conclut qu'il y a moyen de le calculer en plusieurs requêtes mais pour un coût en temps important par rapport à ce qui est demandé. Il est aussi possible de le calculer à partir d'une requête paramétrée.

5.3.4.4 Médiane et mode

À ma connaissance, Rasdaman n'offre pas la possibilité de calculer ces statistiques, même par des moyens détournés comme nous avons essayé de mettre en place pour l'écart type. Par conséquent, des opérations telles que les filtres modaux et médians ne sont pas disponibles au sein de Rasdaman.

5.3.5 Les opérations binaires entre MDD

Ici, nous allons illustrer les opérations binaires entre MDD en calculant un MDD moyen de deux autres. Nous allons calculer la moyenne entre deux pixels homologues sur toute l'image. Nous remettrons ensuite le résultat dans l'encodage de départ en arrondissant. La syntaxe est la suivante :

```
select tiff((char)((((unsigned short)a+
     (unsigned short)b)/2d)+0.5d))
from stopcol as a, spotcol as b
where oid(a)=15361 and oid(b)=15872
```

Attention à ne pas oublier les changements de types sinon le résultat risque de ne pas être celui attendu. Le résultat a été validé en effectuant le même traitement via le logiciel Idrisi. Les performances sont très bonnes :

Temps [s]	Opérations binaires
Minimum	0.44
Maximum	0.61
Moyenne	0.48
Écart type	0.04

5.3.6 Masquage binaire

Nous nous intéressons maintenant à la construction d'un masque binaire (Donnay et Cornet 2009). Il se construit sur base d'une condition. Les cellules prennent la valeur de « 1 » là où la condition est remplie et la valeur de « 0 » ailleurs. Nous allons ici créer un masque d'altitude grâce à la collection `mns`. Si on prend comme condition que les cellules valent strictement plus que 100 (mètres), la requête se formule :

```
select tiff(mdd > 100)
from mns as mdd
```

La requête est donc toute simple. De plus, Rasdaman offre un élégant moyen d'appliquer le masque sur l'image de départ en un seul passage :

```
select tiff(mdd * (mdd > 100))
from mns as mdd
```

Les performances de ces deux requêtes sont excellentes :

Temps [s]	Création du masque	Création et application du masque
Minimum	0.26	0.5
Maximum	0.33	0.63
Moyenne	0.28	0.53
Écart type	0.02	0.02

La création et l'application du masque nécessite environ deux fois plus de temps que sa création seule. Les opérations sont néanmoins très rapides. Elles ont pu être validées grâce à l'exécution des mêmes requêtes via le logiciel Idrisi. Une illustration des masques est fournie à la figure 5.3.

En revanche, une approche plus orientée requête et qui voudrait extraire la liste des coordonnées pour lesquelles le masque vaut « 1 » n'est pas possible; Rasdaman n'offre pas de mécanismes pour extraire des coordonnées.

5.3.7 Reclassification

La reclassification est assez similaire au masque, il s'agit de créer un MDD dérivé contenant plusieurs classes. Par exemple, si on veut créer une classification du modèle de surface par tranches de 100m, la requête suivante est appropriée :

```
select tiff(
     (mdd =>100 and mdd < 200)
     + (mdd => 200 and < 300) * 2c
from mns as mdd
```

Notons qu'on crée trois classes d'altitude : de 0 à 100m et plus de 300m, de 100m à 200m et de 200 à 300m. En fait, on crée deux MDD binaires intermédiaires, le premier pour la classe d'altitudes de 100m à 200m. Toutes les cellules appartenant à cette classe prennent la valeur de « 1 » dans ce MDD intermédiaire. Le second MDD binaire prend la valeur de « 1 » pour toutes les cellules appartenant à 200-300m et « 0 » ailleurs. Ce MDD est alors multiplié par deux. On additionne ensuite les deux MDD. Le résultat est un troisième MDD avec trois valeurs. La valeur zéro dénote toutes les cellules qui sont inférieures à 100m et supérieures à 300m. Cette classe est satisfaisante puisqu'il n'y a pas dans le modèle de valeur supérieure à 300. La première catégorie, créée par défaut, est donc la classe d'altitudes inférieures à 100m. La valeur « 1 » est attribuée aux cellules dont la valeur allait de 100m à 200m et la valeur « 2 » de 200m à 300m. Vu que les tranches d'altitudes sont exclusives les unes par rapport aux autres, aucune nouvelle cellule ne peut changer de classe par addition des MDD intermédiaires.

Une fois encore, ce traitement a été validé grâce au logiciel Idrisi. Une illustration de la reclassification est présentée à la figure 5.4 et les performances sont les suivantes :

Temps [s]	Reclassification
Minimum	0.37
Maximum	1.31
Moyenne	0.40
Écart type	0.09

La valeur maximale est aberrante et ne s'est présentée qu'une seule fois sur les cent mesures. Si on l'enlève, on retombe à un maximum de 0.48. L'opération semble plus coûteuse qu'un simple masque mais moins que la création et l'application de celui-ci.

Rappelons que c'est la connaissance des métadonnées par l'utilisateur qui permet de savoir que l'altitude est exprimée en mètres et que les valeurs ne dépassent pas 300 mètres. Si cette dernière particularité était ignorée, il faudrait adopter une requête plus prudente :

```
select tiff(
     (mdd < 100)
```

```
        (mdd =>100 and mdd < 200)* 2c
        + (mdd => 200 and < 300) * 3c
    from mns as mdd
```

La première classe (valeur « 0 ») reprendrait alors toutes les cellules supérieures à 300 mètres. La deuxième classe (valeur « 1 ») contiendrait les cellules dont l'altitude est strictement inférieure à 100m. Les deux dernières classes de cette requête restent identiques à celles de la requête précédente.

Enfin, on pourrait laisser la dernière classe ouverte :

```
select tiff(
     (mdd =>100 and mdd < 200)
     + (mdd => 200) * 2c
from mns as mdd
```

La valeur de « 0 » témoignerait alors des altitudes supérieures à 200m.

5.3.8 Mosaïquage

Le mosaïquage consiste à reconstruire une grande image à partir d'images plus petites en les assemblant. En général, la reconstruction des images est automatique via les coordonnées géographiques des coins de l'image. Rasdaman ne gérant pas les coordonnées géographiques, il faut se charger manuellement du positionnement des images. De plus, ces données doivent être extraites des métadonnées des images géographiques de départ. Le mosaïquage n'est donc pas l'opération emblématique de Rasdaman. Néanmoins, disposant des bonnes informations, il est possible de réaliser le mosaïquage facilement.

Si on regarde les métadonnées, on s'aperçoit que la première image Landsat est située au nord de la seconde et qu'il existe un recouvrement des 100 premières lignes de l'image. Sachant cela, la requête de mosaïquage pour les deux MDD correspondants s'exprime :

```
select tiff(
    extend(a, [0:1023, 0:1099]) overlay
        extend(shift(b,[0,500]),[0:1023, 0:1099]))
from landsatcol as a, landsatcol as b
where oid(a)=13313 and oid(b)=13825
```

Le MDD « a » correspond à l'image au nord et le MDD « b » correspond à l'image au sud. Les deux MDD ont pour domaine spatial de départ $[0:1023, 0:599]$. Il faut donc d'abord décaller le MDD « b » de 500 cellules vers le sud afin que les MDD soient bien positionnés l'un par rapport à l'autre. Ceci s'effectue grâce à la fonction `shift`() qui prend comme arguments le MDD à déplacer ainsi que le vecteur de translation. Ensuite, il faut étendre les deux domaines spatiaux pour qu'ils s'intersectent pleinement. Cette opération va introduire des valeurs nulles en dehors de l'ancien domaine spatial. Ces valeurs nulles vont être pratiques pour utiliser l'opérateur `overlay` qui vient superposer le premier opérande au-dessus du second sauf là où les valeurs sont nulles. Le résultat du mosaïquage est montré à la figure 5.5.

En toute logique, les deux MDD devraient avoir leur domaine spatial cohérent dès le début. C'est à dire, le MDD « b » devrait avoir un domaine spatial translaté dès le départ : $[0:1023, 500:1099]$. Il serait alors aisé d'appliquer

la requête sans devoir aller dans les métadonnées des images de départ. Néanmoins, une mise à jour du domaine spatial d'un MDD ne produit pas le résultat attendu. La requête :

```
update landsatcol as mdd
set mdd assign shift(mdd, [0:500])
where oid(mdd)=13825
```

ne se contente pas de déplacer le MDD vers le sud mais étend le domaine spatial du MDD et rajoute la partie déplacée par au-dessus. Le MDD mis à jour a donc un domaine spatial de [0 : 1024, 0 : 1099]. Ceci est une spécificité de la requête qui détermine les cellules à mettre à jour via le domaine spatial du MDD fourni après le mot clé `assign`. Rasdaman va donc étendre le MDD pour y rajouter celui qu'on lui fournit.

Cette particularité implique qu'il n'est pas possible de changer le domaine spatial des MDD. Certes, celui-ci ne peut pas s'établir sur des nombres décimaux et ne peut donc reprendre les coordonnées géographiques (sans leur système de référence associé) mais translater les MDD aurait pu permettre de les situer les uns par rapport aux autres.

L'opération de mosaïquage est donc possible pour autant que les métadonnées des images de départ soient connues. Les performances de l'opération sont cependant excellentes (statistique sur 100 observations) :

Temps [s]	Changement d'échelle
Minimum	0.44
Maximum	0.75
Moyenne	0.49
Écart type	0.05

5.3.9 Changements d'échelle et interpolation

Nous allons maintenant doubler le domaine spatial du MDD grâce à la fonction `scale`() sur la collection `landsatcol`. La requête pour extraire les MDD sous forme d'images TIFF est la suivante :

```
select tiff(
    scale(mdd, 2))
from landsatcol as mdd
```

On fournit donc le MDD ainsi que le coefficient d'expansion. La requête est modérément rapide (statistiques sur 100 observations) :

Temps [s]	Changement d'échelle
Minimum	1.84
Maximum	2.98
Moyenne	1.95
Écart type	0.23

Le domaine spatial du nouvel MDD peut être examiné par

```
select sdom(scale(mdd, 2))
from landsatcol as mdd
```

et vaut [0 :2047, 0 :1199]. Il s'agit en fait d'un zoom comme on peut le tester via le couple de requêtes :

```
select mdd[0,0] from landsatcol as mdd
select scale(mdd, 2)[1,1] from landsatcol as mdd
```

et en faisant varier les cellules sélectionnées. C'est aussi précisément ce qui se passe quand on effectue le même traitement sur un MDD codé en réels. Il n'y a donc aucune interpolation qui se produit si la même requête est effectuée sur la collection `mns`. En fait, Rasdaman n'offre de base aucun mécanisme pour interpoler entre cellules. Il faut donc l'implémenter. Par exemple, si on veut effectuer une interpolation bilinéaire aux coordonnées (1.4, 1.7) :

```
select tiff(0.3*(0.6*mdd[1,1]+0.4*mdd[1,2])
    +0.7*(0.6*mdd[2,1]+0.4*mdd[2,2]))
from mns as mdd
```

Ce qui demande la paramétrisation pour chaque interpolation. Par contre, on peut effectuer la même interpolation (aux coordonnées (x.4, y.7)) sur toutes les cellules du MDD :

```
select tiff(
    marray x in [sdom(mdd)[0].lo : sdom(mdd)[0].hi - 1,
        sdom(mdd)[1].lo : sdom(mdd)[1].hi - 1]
    values 0.3*(0.6 * mdd[x[0], x[1]] + 0.4 * mdd[x[0], x[1]] +1)
        +0.7*(0.6 * mdd[x[0]+1, x[1]] + 0.4 * mdd[x[0]+1, x[1]+1]))
from mns as mdd
```

L'instruction `sdom()` renvoie le domaine spatial du MDD, comme à l'accoutumée. L'instruction `sdom(mdd)[0]` sélectionne la première composante du domaine spatial. Quant à l'instruction `sdom(mdd)[0].lo`, elle extrait la borne inférieure de la première dimension du domaine spatial. On crée ainsi un nouveau MDD dont le domaine spatial est inférieur d'une unité dans chaque dimension. Pour chaque cellule du nouvel MDD, on applique l'interpolation bilinéaire sur les cellules correspondantes de l'ancien.

S'il avait été possible de définir ses propres fonctions, on aurait facilement pu en créer une pour l'interpolation bilinéaire.

5.3.10 Amélioration de constraste

Le présent traitement a pour but d'améliorer le contraste d'un MDD. Il s'agit de redistribuer les valeurs des cellules du MDD pour qu'elles occupent l'entiereté de l'intervalle de valeur disponible. Nous allons utiliser la collection `spotcol`. L'intervalle de valeurs disponibles sera le même que ce que la collection permet, c'est-à-dire [0 : 255]. Nous allons ici tester une amélioration linéaire (Donnay et Cornet 2008) :

$$x' = (255 - 0) \cdot \frac{x - min}{(max - min)} + 0$$

La nouvelle valeur de la cellule, x', est obtenue par interpolation linéaire à partir de l'ancienne, x. min et max représentent respectivement le minimum et le maximum dans le MDD de départ.

Testons la requête sur le premier MDD de la collection `spotcol` dont le minimum vaut 14 et le maximum 194 :

```
select tiff(
    marray x in sdom(mdd)
        values (char)(((mdd[x[0], x[1]]) - 14)/180d)*255))
from spotcol as mdd
where oid(mdd)=15361
```

On construit un nouveau MDD dont le domaine spatial est le même que l'ancien. La variable `x` sert d'itérateur sur le domaine. `x[0]` fait référence à la première dimension et `x[1]` à la seconde. L'instruction `mdd[x[0], x[1]]` signifie donc qu'on va lire la valeur de chaque cellule du MDD représenté par `mdd`. Afin de réaliser une division conventionnelle et non une division entière, il faut préciser que la valeur `180` qui représente l'étendue est un réel. Enfin, on remet le résultat dans le bon intervalle. Dernière remarque : il faut bien veiller à laisser un espace avant et après le signe négatif, Rasdaman est pointilleux sur le sujet.

L'amélioration de contraste est une tâche coûteuse, comme le montre le tableau suivant (statistiques sur 100 observations) :

Temps [s]	Amélioration d'histogramme
Minimum	4.54
Maximum	5.73
Moyenne	4.87
Écart type	0.32

La généralisation de la requête pose cependant problème. Tout comme pour l'écart type, il s'agit d'imbriquer des fonctions d'agrégation dans des boucles sur tout le domaine spatial. En particulier, les fonctions `min_cells`() et `max_cells`(). Ceci amène à une trop grosse consommation de ressources. La requête serait :

```
select tiff(
    marray x in sdom(mdd)
         values (char)(((mdd[x[0], x[1]]) - min_cells(mdd))
             /(double)(max_cells(mdd)-min_cells(mdd))*255))
from spotcol as mdd
where oid(mdd)=15361
```

Dès lors, il est possible de calculer une amélioration de contraste mais pas de manière automatique. Il faut d'abord extraire les statistiques du MDD puis paramétrer la requête.

5.3.11 Histogrammes

Nous allons à présent nous intéresser au calcul d'histogrammes. Dans un premier temps, nous discuterons du cas univarié avant de passer à celui d'un histogramme à deux variables.

5.3.11.1 Histogramme univarié

Nous allons tester le calcul d'histogrammes sur la collection `landsatcol`. Pour ce faire, on doit créer un nouveau MDD 1D, itérer sur son domaine et utiliser une fonction `count_cells`(). Il s'agit donc d'utiliser une fonction agrégative dans une boucle. Précédemment, cela avait posé problème (le calcul de l'écart type). Néanmoins, ici on travaille sur un MDD unidimensionnel. Pour

calculer l'histogramme sur les valeurs de 0 à 255 avec des classes unitaires, la syntaxe est la suivante :

```
select
    marray x in [0:255]
        values count_cells(mdd=x)
from landsatcol as mdd
```

Cette requête fonctionne mais Rasdaman n'offre que la possibilité de l'extraire sous forme hexadécimale...

```
rasql -q 'select marray x in [0:255]
    values count_cells(mdd=x) from landsatcol as mdd'
        --out hex
```

Si on veut créer un histogramme à classe unitaire, commençant au minimum et terminant au maximum, il faut adopter la requête :

```
select
    marray x in [ min_cells(mdd):max_cells(mdd) ]
         values count_cells(mdd=x)
from landsatcol as mdd
```

Finalement, si on veut réaliser un histogramme avec des classes non-unitaires (mais de tailles constantes), par exemple des classes de taille 5 avec la borne inférieure exclue et la borne supérieure inclue, il faut utiliser la requête :

```
select
   marray x in [0:50]
        values count_cells(mdd > (5c * x)
            and mdd <= 5c * (x+1))
from landsatcol as mdd
```

Après conversion des valeurs hexadécimales en décimales, il est possible de vérifier le résultat grâce au module `gdalinfo` de GDAL, par exemple. Celui-ci est correct et les performances des deux premières requêtes sont les suivantes (respectivement sur 100 et 30 observations) :

Temps [s]	bornes fixées : [0 :255]	bornes minimales et maximales
Minimum	18.34	18.57
Maximum	31.82	20.74
Moyenne	19.22	18.97
Écart type	1.32	0.39

Ces quelques statistiques sont intéressantes. Notons d'abord que la valeur de 31.82 est une valeur aberrante qui ne s'est présentée qu'une seule fois sur l'échantillon de cent requêtes. Si on la supprime, le maximum passe à 21.01. Le nombre de classes sur lequel on calcule l'histogramme a un impact sur les performances. Par exemple, dix classes ne prendront que 0.94 s en moyenne. Cependant, ceci n'entre pas en compte ici puisqu'il n'y a qu'une classe de différence (la valeur minimale de cellule vaut « 1 » et non « 0 »). On remarque que le surcoût pour calculer le minimum et maximum est négligeable quand il est en dehors des itérations. L'opération coûteuse est donc l'évaluation de la fonction `count_cells`() qui doit se faire pour chaque classe de l'histogramme. Néanmoins, pour de petits histogrammes, la consommation de ressources ne semble

pas être trop importante. Ainsi, s'il faut compter une vingtaine de secondes pour calculer un histogramme de 255 classes, la requête arrive à terme. Ce qui n'était pas le cas des requêtes de calcul d'écart type.

5.3.11.2 Histogramme bivarié

Par histogramme à deux variables, il faut comprendre qu'on comptabilise pour chaque classe de la variable a et pour chaque classe de la variable b, l'occurrence de se trouver dans la ième classe de a et jième classe de b en même temps. Dans le cas présent, ça signifie compter le nombre de cellules du premier MDD qui ont une certaine valeur pendant que les cellules du second MDD ont une autre valeur déterminée. Cet histogramme peut être élégamment présenté sous la forme d'une image (Donnay et Cornet 2008). La syntaxe d'une telle réalisation est, pour la collection `spotcol` :

```
select tiff(
    marray x in [0:255, 0:255]
        values count_cells (a=x[0] and b=x[1]))
from spotcol as a, spotcol as b
where oid(a)=15361 and oid(b)=15873
```

Comme on peut le constater il s'agit d'exécuter une fonction `count_cells`() au sein d'une boucle de 65536 itérations. Cette requête consomme trop de ressources. Nous avons vu qu'il était possible d'aller au moins jusqu'à 256 itérations pour les histogrammes univarié. Avec un peu de patience (plus de huit minutes ; 518.68 secondes), j'ai pu calculer un histogramme de 75x75 cellules. L'opération est donc coûteuse. À titre informatif, des statistiques ont été calculées sur des échantillons de 50 requêtes pour des histogrammes de 11x11, 26x26 et 51x51 cellules :

Temps [s]	11x11 cellules	26x26 cellules	51x51 cellules
Minimum	9.74	53.3	192.88
Maximum	12.02	71.78	196.19
Moyenne	10.4	56.35	193.65
Écart type	0.73	3.86	0.65

Il est possible de comparer ces performances avec celles de la collection `spot3` pour laquelle la requête devient :

```
select tiff(
    marray x in [0:10, 0:10]
        values (char)count_cells (a.0=x[0] and a.1=x[1]))
from spot3 as a
```

Au niveau de la requête, seule la manière de sélectionner les bonnes couches d'information change. Par contre, les performances sont moins « bonnes » qu'auparavent comme le montre le tableau suivant (statistiques sur 50 observations) :

Temps [s]	11x11 cellules	26x26 cellules	51x51 cellules
Minimum	12.68	71.06	256.86
Maximum	13.59	74.15	297.88
Moyenne	12.91	72.10	264.91
Écart type	0.17	0.62	10.25

Pour conclure, on peut dire que la création d'histogrammes bivariés est une opération coûteuse et ne fonctionne que pour de petits MDD. De plus, la répartition des différentes couches d'information en plusieurs MDD au sein d'une collection offre de meilleurs résultats que l'agglomération de ces couches au sein d'un seul MDD composé. Ceci est avantageux puisque Rasdaman n'offre pas de mécanisme pour construire facilement cette dernière catégorie de MDD.

5.3.12 Les fenêtres de convolution

Rasdaman permet l'expression des opérations de convolution (Donnay et Cornet 2008) via le constructeur de MDD `marray` et l'opérateur `condense`. La syntaxe générale est la suivante (pour une fenêtre de convolution de taille 3x3) :

```
marray x in [sdom(mdd)[0].lo + 1: sdom(mdd)[0].hi - 1,
    sdom(mdd)[1].lo + 1: sdom(mdd)[1].hi - 1]
values condense +
over y in sdom(filtre)
using (mdd[ x[0] + y[0], x[1] + y[1] * filtre[y])
```

où `mdd` est le MDD sur lequel on réalise l'opération et `filtre` est un MDD contenant le filtre et dont le domaine spatial est centré sur (0,0).

Expliquons cette requête. On crée un MDD `x` dont le domaine spatial est plus petit que le MDD sur lequel on applique le filtre. Il est plus petit de deux rangées selon la première dimension : la première rangée (`sdom(mdd)[0].lo + 1`) et la dernière (`sdom(mdd)[0].hi − 1`). Il est aussi plus petit de deux rangées selon la deuxième dimension, à nouveau la première et la dernière rangée. Ceci signifie que la fenêtre de convolution ne se centrera sur aucune des cellules formant le bord du MDD. Cette condition est nécessaire pour que la fenêtre ne sorte pas du domaine spatial du MDD originel.

Pour construire le nouveau MDD (`x`), on doit procéder à une moyenne pondérée des pixels dans la fenêtre de convolution. L'opérateur `condense` + prépare donc la somme des termes. Le nombre de termes est défini par `y` qui est un itérateur sur le domaine spatial du filtre. Le calcul des termes est défini par ce qui suit le mot clé `using`. `mdd`[...] va rechercher les valeurs du MDD dans la fenêtre de convolution. Enfin, `m`[`y`] définit le coefficient de pondération.

Il y a donc deux boucles. La première parcourt le MDD original et la seconde la fenêtre de convolution.

5.3.12.1 Filtrage moyen

Ainsi, si on veut filtrer le modèle numérique par la moyenne avec une fenêtre de convolution de taille 3x3, le filtre est :

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

qu'il faut mettre dans la base de données :

```
create collection filtre_moy FloatSet
insert into filtre_moy values
    marray x in [-1:1, -1:1] values 1/9d
```

Le filtrage s'exprime alors :

```
select tiff(
    marray x in [sdom(mdd)[0].lo + 1: sdom(mdd)[0].hi - 1,
        sdom(mdd)[1].lo + 1: sdom(mdd)[1].hi - 1]
    values condense +
    over y in sdom(filtre)
    using (mdd[ x[0] + y[0], x[1] + y[1] * filtre[y]))
from mns as mdd, filtre_moy as filtre
```

Notons cependant que pour le filtrage par moyenne, il est possible de faire plus simple grâce à la fonction `avg_cells`() :

```
select tiff(
    marray x in [sdom(mdd)[0].lo + 1: sdom(mdd)[0].hi - 1,
        sdom(mdd)[1].lo + 1: sdom(mdd)[1].hi - 1]
    values avg_cells(mdd[ x[0] - 1 + x[0] +1,
        x[1] - 1 + y[1] +1]))
from mns as mdd
```

Contre toutes attentes, la première requête est plus performante que la seconde, comme l'illustrent ces statistiques sur 50 observations :

Temps [s]	Condenseur	`avg_cells`()
Minimum	10.29	21.68
Maximum	11.05	22.85
Moyenne	10.51	22.18
Écart type	0.15	0.23

5.3.12.2 Calcul de pentes

Nous allons maintenant calculer des pentes à partir du modèle numérique de surface. Appelons x la première dimension, y la deuxième et z l'altitude. Toutes les unités sont des mètres. Le calcul de pentes est un traitement compliqué. Il nécessite le calcul de deux filtres ainsi que l'application de plusieurs opérations. Pour une pente en degré (Donnay 2008) :

$$pente\,(x,y) = \frac{180}{\pi} \arctan \sqrt{\left(\overrightarrow{\nabla}_x z\right)^2 + \left(\overrightarrow{\nabla}_y z\right)^2}$$

Les gradients discrets $(\overrightarrow{\nabla}.z)$ selon x et y peuvent être obtenus respectivement à partir des filtres de Sobel (Donnay et Cornet 2008) (qu'il faut normaliser en divisant par 8) :

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

Ces filtres peuvent être introduits dans la base de données :

```
create collection gradx FloatSet
create collection grady FloatSet
insert into collection gradx values <[-1:1, -1:1]
    -0.125, -0.25, -0.125;0.0, 0.0, 0.0;
```

```
     0.125, 0.25, 0.125>
insert into collection grady values <[-1:1, -1:1]
    0.125, 0.0, -0.125;-0.25, 0.0, -0.25;
    0.125, 0.0, -0.125>
```

Avant de calculer les pentes, regardons déjà la requête jusque la somme des carrés des gradients :

```
select tiff(
    marray x in [sdom(mdd)[0].lo + 1: sdom(mdd)[0].hi - 1,
        sdom(mdd)[1].lo + 1: sdom(mdd)[1].hi - 1]
    values condense +
    over y in sdom(gx)
    using ((mdd[ x[0] + y[0], x[1] + y[1] * gx[y])
        *(mdd[ x[0] + y[0], x[1] + y[1] * gx[y])
        +(mdd[ x[0] + y[0], x[1] + y[1] * gy[y])
        *(mdd[ x[0] + y[0], x[1] + y[1] * gy[y])))
from mns as mdd, gradx as gx, grady as gy
```

On calcule donc les deux gradients, qu'on met au carré et qu'on additionne. Il manque encore l'application de la racine carrée, de la fonction arctangeante et de la conversion des radians en degré. Rasdaman ne fournit pas de constantes ou fonctions toute faites pour π. Il faut soit fournir une valeur approchée, soit utiliser la fonction arcosinus en -1 pour l'obtenir. La requête complète du calcul de pente est donc :

```
select tiff((180/arccos(-1))*arctan(sqrt(
    marray x in [sdom(mdd)[0].lo + 1: sdom(mdd)[0].hi - 1,
         sdom(mdd)[1].lo + 1: sdom(mdd)[1].hi - 1]
    values condense +
    over y in sdom(gx)
    using (mdd[ x[0] + y[0], x[1] + y[1] * gx[y])
        *(mdd[ x[0] + y[0], x[1] + y[1] * gx[y])
        +(mdd[ x[0] + y[0], x[1] + y[1] * gy[y])
        *(mdd[ x[0] + y[0], x[1] + y[1] * gy[y]))))
from mns as mdd, gradx as gx, grady as gy
```

Le calcul de pentes a pu être validé grâce au logiciel Idrisi. Les performances, calculées sur 50 observations sont les suivantes :

Temps [s]	Amélioration d'histogramme
Minimum	195.73
Maximum	200.89
Moyenne	197.50
Écart type	0.95

L'opération dure un temps certains. Plus de trois minutes semlbe un temps prohibitif malgré la complexité de l'opération. Cependant, cette requête n'a pas besoin d'être paramêtrée, elle est valable pour tous les modèles numériques.

5.3.13 Requêtes sur les MDD

Certains traitements peuvent s'apparenter à des requêtes, comme c'est le cas des masques binaires. Les métadonnées n'étant pas disponibles, toutes les

requêtes se basant sur elles ne sont pas envisageables. Une requête du style « Quels sont les MDD qui ont été acquis entre le 10 janvier 2011 et le 10 août 2011 ? » n'est pas possible. Les seules requêtes possibles sont celles qui ne traitent que des valeurs des MDD. Soulignons qu'une requête telle que « Quels sont les MDD dont la couverture neigeuse dépasse 50% ? » n'est pas non plus possible car il n'y a pas de métadonnées pour documenter ce qu'est une couverture neigeuse ; même si le MDD est nominal et que la couverture neigeuse correspond à la valeur 4, par exemple.

Dans cette section, nous allons tester deux requêtes ne se basant pas sur les métadonnées. La première est :

> « Combien de cellules du MNS ont une valeur supérieure à 100 ? »

Cette requête se rapproche fortement de la dérivation d'un masque binaire et s'exprime comme suit :

```
select count_cells( mdd > 100 )from mns as mdd
```

Attention, la sortie est un nombre et non pas un MDD, il faut donc adapter les paramètres du module `rasql`. Elle a été validée grâce au logiciel Idrisi. Il est évident que, disposant des métadonnées, la requête aurait pu se poser en terme de superficie et préciser qu'il s'agissait de 100 mètres. La deuxième requête est :

> « Quels MDD de la collection `spotcol` possèdent parmi leurs cellules une valeur inférieure à 20 ? »

Nous allons ici extraire l'OID du MDD répondant à la condition :

```
select oid(mdd)
from spotcol as mdd
where some_cells(mdd < 20 )
```

Remarque intéressante : la configuration de la collection `spot3` ne permet pas de faire cette requête. En effet, il n'y a pas une fonction comme `oid`() qui permet d'extraire le numéro de la composante.

Les performances de ces deux requêtes sont excellentes, comme on pouvait s'y attendre. Les statistiques suivantes ont été calculées sur un échantillon de 100 observations :

Temps [s]	Première requête	Deuxième requête
Minimum	0.14	0.23
Maximum	0.27	0.50
Moyenne	0.16	0.26
Écart type	0.03	0.05

La seconde requête est plus coûteuse que la première car elle doit parcourir quatre MDD de 614400 cellules au lieu d'un. Si les deux requêtes devaient parcourir le même nombre de cellules, la seconde requête serait évidemment plus efficace (elle ne doit parcourir le MDD que jusqu'à rencontrer un « 1 »).

On peut conclure que Rasdaman effectue efficacement les requêtes qu'il est capable d'accomplir.

5.3.14 Le bilan des traitements et requêtes

Le prototypage des traitements et des requêtes a permis de tester les capacités de Rasdaman. Le bilan est mitigé. Les requêtes testées fonctionnent mais

le champ des requêtes a été considérablement réduit par l'absence des métadonnées. En ce qui concerne les traitements, beaucoup sont possibles. Parmi ceux-ci un bon nombre sont performants : le calcul du minimum, du maximum, de la moyenne, les reclassifications, le mosaïquage, et les changements d'échelle.

D'autres sont malheureusement coûteux : l'amélioration de contraste paramêtrée et les histogrammes en particulier. Plus généralement, tous les traitements utilisant la construction de nouveaux MDD de taille importante sont coûteux.

De plus, l'imbrication d'itérateurs sur de gros domaines spatiaux amène souvent à un coût prohibitif voir une trop grosse consommation de ressources. C'est notamment le cas du calcul de l'écart type et des améliorations de contrastes non paramétrées. On regrette donc que l'optimisateur de Rasdaman ne soit pas capable de déceler les valeurs qui restent constantes et qui ne devraient être calculées qu'une seule fois. D'autant plus, quand il s'agit du maximum et minimum qui sont d'habitude présents dans les métadonnées. L'alternative pour faire redescendre le coût des requêtes est la paramétrisation, qui suppose l'extraction préalable d'informations. Une requête qui pourrait être automatique doit donc s'effectuer en plusieurs temps.

La définition de fonctions par l'utilisateur pourrait refaire passer ces traitements en une seule requête grâce au jeu de variables mais ceci n'est pas permis par Rasdaman. La définition de fonctions utilisateurs permettrait également l'enrichissement du langage avec des fonctions d'interpolations, notamment.

Dans les autres caractéristiques manquantes du langage, on peut noter des fonctions permettant de calculer le mode et la médiane des valeurs des cellules.

5.4 Confrontation partielle aux critères de validation

Les résumés des prototypages des données, des traitements et requêtes ayant déjà été réalisés, nous allons directement passer à la confrontation de ce que nous avons découvert et des critères de validation.

Au niveau des fonctionnalités, nous avons juste mis en évidence le problème du catalogue de données. Rasdaman n'offre pas vraiment cette capacité. Le plus proche est une liste des types (cellules, MDD, ensembles) définis et des collections. Mais aucune description n'accompagne ces données.

Au niveau des données, nous avons pu voir que, si Rasdaman sait gérer un grand type de données, l'insertion d'images sous forme de MDD composés et sous forme d'images 3D n'est pas bien supportée. Ces types de données sont donc inutilisables. Néanmoins, il est possible d'insérer des images sous forme de MDD individuels dans une collection. Cette forme de stockage permet d'ailleurs de meilleures performances pour les applications que nous avons étudiées.

Au niveau des traitements, nous avons constaté qu'un bon nombre d'entre eux étaient réalisés rapidement par Rasdaman. D'autres, en revanche, étaient peu performants. C'est le cas des traitements utilisant plusieurs itérateurs. Finalement, certains traitements sont impossibles : la création de composition colorée, les calculs de mode et de médiane, notamment.

(a) Le masque seul

(b) Le masque appliqué au modèle numérique de surface

FIGURE 5.3 – Illustration de l'opération de masquage binaire

FIGURE 5.4 – Illustration de la reclassification (après adaptation de la palette)

S

FIGURE 5.5 – Le résultat du mosaïquage

Chapitre 6

Résultat de la recherche

6.1 Confrontation de l'hypothèse

Nous allons maintenant prendre le temps de récapituler ce qui a été découvert au cours de ce mémoire. Pour cela, nous allons nous appuyer sur les confrontations partielles aux critères de validation. Les critères seront passés en revue et puis nous statuerons sur l'hypothèse.

Les critères de validation étaient organisés en trois parties : fonctionnalité, données et, traitements et requêtes. Nous allons reprendre chaque catégorie séparément.

6.1.1 Les fonctionnalités

Parmi les fonctionnalités figuraient les mécanismes d'insertion et d'extraction depuis et vers les formats les plus connus. Ce premier point n'est pas totalement rempli. En effet, les formats pris en charge sont principalement des formats images. Certes, Rasdaman gère le TIFF, l'HDF 4 et le DEM mais nombre de formats géographiques ne sont pas supportés. De plus, Rasdaman ne tient pas compte des métadonnées géographiques. On conclut que le critère n'est pas rempli de manière satisfaisante.

La seconde fonctionnalité était la disponibilité d'un langage de requête. Un tel langage est bel et bien présent mais il ne permet pas d'interroger les métadonnées (dans l'hypothèse où celles-ci auraient été rajoutées). Néanmoins, un critère ultérieur viendra juger cela. En ce qui concerne la présence du langage de requête, la condition est remplie.

La troisième fonctionnalité était la construction d'un catalogue de données. Cette dernière est très basique et se limite à la liste des types créés et des collections. Aucune autre information n'est disponible, pas même des informations gérées par Rasdaman telles que le type d'ensemble des collections. La condition n'est donc pas remplie.

La quatrième fonctionnalité était l'indexation spatiale. Celle-ci est présente bien que peu documentée. La condition est remplie.

Les deux dernières fonctionnalités étaient optionnelles. Il s'agissait d'offrir la possibilité de créer un serveur web et de mettre à disposition des outils facilitant

la conception d'un serveur de données.

Rasdaman possède PetaScope, lui permettant de créer des serveurs web. Malheureusement, je n'ai pas réussi à le faire fonctionner. Le point ne peut donc pas être évalué.

En revanche, il est possible d'évaluer la dernière fonctionnalité. Rasdaman n'offre pas d'outils facilitant la conception. Ceux-ci ne sont cependant peut-être pas nécessaires vu la manière dont Rasdaman gère les données. La condition n'est pas remplie.

6.1.2 Les données

Parmi les critères de validation en rapport avec les données, Rasdaman devait être capable de gérer les métadonnées. Cette condition n'est pas du tout satisfaite. C'est d'ailleurs l'élément le plus insatisfaisant de Rasdaman car il a des répercussions très importantes : les informations sont perdues lors de l'insertion, le langage de requête ne sait pas interroger sur les métadonnées et certains traitements sont inefficaces par manque de disponibilité de ces informations. Il en va bien sûr de même pour les critères optionnels concernant la gestion des métadonnées de la légende et des couleurs indexées.

Le second critère de cette catégorie était la gestion d'un grand nombre de données différentes. Nous avons pu mettre en évidence que le stockage des *rasters* 2D sous forme de MDD individuels était bien géré par Rasdaman, et ce, quel que soit le type de données. Par contre, la construction de structures plus complexes à partir de *rasters* individuels ne donne pas de bons résultats. Néanmoins, nous avons été capables de stocker tous les MDD dans la base de données. La condition est donc remplie.

Le critère suivant était optionnel et concernait l'intégration de données vectorielles. Nous avons conclu que Rasdaman n'offrait pas de possibilités allant dans ce sens puisque rien n'était prévu dans le langage de requêtes à ce sujet. Le critère est donc insatisfait.

Le dernier critère, optionnel, stipulait une gestion interne simple des données. Nous avons eu le loisir de constater que ce n'était pas le cas. À nouveau, le critère est insatisfait.

6.1.3 Les traitements et requêtes

Sur le plan des requêtes, Rasdaman devait fournir deux choses : la possibilité d'interroger les métadonnées et éventuellement la possibilité d'effectuer des requêtes directement sur les cellules. La première partie n'est pas remplie du tout. En revanche, la seconde partie l'est et Rasdaman est très performant à ce niveau-là.

En ce qui concerne les traitements, la conclusion est mitigée. La création de compositions colorées et les projections ne sont pas supportées. Le mosaïquage n'est possible que si on connaît la position relative des images, auquel cas, il est performant. Certains traitements sont trop coûteux pour pouvoir être validés. C'est le cas de l'amélioration de contraste, des histogrammes et du calcul de pentes ainsi que des traitements non paramétrés (écart type et amélioration de contraste). En revanche, beaucoup de traitements sont satisfaisants : le calcul du minimum, maximum et de la moyenne, les opérations binaires, les reclassifications, les changements d'échelle voir même les fenêtres de convolution simples.

Si on reprend la liste des critères, sont évalués satisfaisants les traitements suivants : le calcul de statistique (pour le minimum et maximum), le changement de résolution spatiale, les opérations binaires, les masquages, les reclassifications et les histogrammes univariés. Les traitements insatisfaisants sont : les reprojections, le mosaïquage, les histogrammes bivariés, l'amélioration de contraste, le calcul de statistique (pour le mode, la médiane et l'écart type) et les compositions colorées. Les fenêtres de convolution simples sont admissibles mais il est impossible d'appliquer des filtres modaux ou médians. De plus, le calcul de pentes n'est pas performant. L'évaluation est donc mitigée sur ce point.

6.1.4 L'hypothèse

Les critères de validation obligatoires ainsi que leur évaluation sont repris au tableau 6.1. Avec celui-ci, nous allons maintenant pouvoir statuer sur l'hypothèse.

Critères	Évaluation
Support des formats géographiques	insatisfaisant
langage de requête	satisfaisant
Catalogue de données	insatisfaisant
Indexation spatiale	satisfaisant
Intégration des métadonnées	**insatisfaisant**
Stockage de différents types de *rasters*	satisfaisant
Requête sur les métadonnées	insatisfaisant
Calcul de statistiques	mitigé
Changement de résolution spatiale	satisfaisant
Opérations binaires	satisfaisant
Reprojection	insatisfaisant
Création de mosaïque	insatisfaisant
Masquages et reclassifications	satisfaisant
Histogrammes et améliorations de contraste	insatisfaisant
Composition colorée	insatisfaisant
Fenêtre de convolution	mitigé

TABLE 6.1 – Les critères indispensables pour valider l'hypothèse et leur évaluation

Pour rappel, l'hypothèse était :

> « Rasdaman, à lui seul, est capable de faire office de serveur de données géographiques maillées, notamment dans le cadre du projet CARE GITAN de l'ULg ».

Le bilan des critères de validation indique que Rasdaman ne remplit que la moitié des critères de validation. Nous ne pouvons donc pas valider l'hypothèse ; Rasdaman, à lui seul, ne permet pas de faire office de serveur de données géographiques maillées.

6.2 Rasdaman comme composant d'une architecture

Rasdaman ne permet pas, à lui seul, de faire office de serveur de données géographiques principalement parce qu'il n'est pas équipé pour gérer les métadonnées géographiques. Dans cette section, nous allons voir comment enrichir Rasdaman afin de combler ses manques. Ce qui va suivre n'est qu'une manière parmi d'autres d'y parvenir.

6.2.1 L'architecture

Trois composants sont nécessaires : PostGIS, Rasdaman et GDAL. L'architecture du nouveau serveur est illustrée à la figure 6.1. Le rôle de Rasdaman reste le même.

PostGIS est une extension de PostgreSQL qui permet de gérer les données géographiques vectorielles (http ://postgis.refractions.net). Le système de gestion de base de données reste donc bien PostgreSQL auquel on rajoute PostGIS. Son rôle dans notre application est double. D'une part, il doit apporter le support des systèmes de référence et d'autre part il va permettre de conserver le cadre capable des *rasters*.

Quant à la couche supplémentaire, c'est elle qui est responsable de la communication entre le client et les serveurs de données (Rasdaman et PostGIS). De plus, cette couche contient les utilitaires de GDAL permettant d'extraire et d'insérer les métadonnées (géographiques).

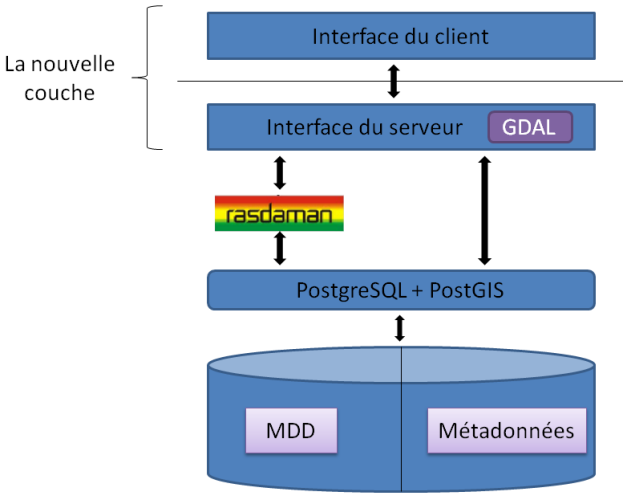


FIGURE 6.1 – Architecture du serveur de données enrichi

Le module `gdalwrap` par exemple, permet d'ajouter un système de références aux *rasters*. Ainsi, il est possible d'extraire de Rasdaman un MDD, de PostGIS son référentiel et de remettre le référentiel dans l'image extraite. GDAL permet aussi la gestion d'un grand nombre de formats géographiques, le calcul de

statistiques, la gestion des couleurs indexées, les projections,...

6.2.2 Les métadonnées

Afin de gérer les métadonnées, il faut rajouter quelques tables. Les tables ci-après ont été imaginées en fonction des hypothèses suivantes :

- ⋆ Tous les MDD d'une même collection doivent avoir le même système de référence.
- ⋆ Les MDD sont 2D.
- ⋆ Les MDD peuvent avoir un type de cellules composé.

Une première table devrait avoir un rapport avec la collection et devrait contenir au minimum trois informations :

```
collection(nom, description, srid)
```

Le nom de la collection sert de clé primaire. La description permettrait d'identifier rapidement ce que contient la collection. Enfin, le `srid` serait une clé étrangère vers les tables de PostGIS contenant les informations des systèmes de référence. Cela permettrait de forcer la contrainte selon laquelle les MDD d'une même collection doivent tous avoir le même système de référence. Il est possible d'extraire, à partir du nom de la collection, le type de MDD qu'elle peut contenir via les tables créées par Rasdaman (*cfr.* section 4.5).

La seconde table serait en rapport avec les MDD. Elle reprendrait tous les MDD entrés dans Rasdaman et aurait la structure suivante :

```
MDD(oid, description, cadre_capable, date, résolution_x,
    résolution_y, unité)
```

L'OID servirait de clé primaire et permettrait de faire le lien avec les tables de Rasdaman. De celui-ci, on pourrait déduire le domaine spatial du MDD. La description permettrait d'identifier rapidement le MDD pour l'utilisateur. Le cadre capable serait un objet vectoriel de PostGIS représentant l'extension du MDD dans le système de référence du *raster*. Il permettrait d'effectuer des requêtes pour savoir, par exemple, si un point donné en coordonnées géographiques appartient au MDD. Les autres métadonnées reprises dans la table sont la date d'acquisition, la résolution selon les deux axes et les unités des axes. L'unité peut être une clé étrangère vers une table statique représentant toutes les unités gérées ou une énumération [1].

La troisième table contiendrait, pour chaque MDD ou chaque composante de MDD, des informations sur les valeurs contenues dans les cellules :

```
range(oid, #composante, unité, minimum, maximum)
```

L'OID permettrait d'identifier le MDD. S'il s'agit d'un MDD à type de cellules composé, l'attribut #`composante` permettrait d'identifier de quelle composante il s'agit. Sinon, il est laissé vide. Les trois attributs suivants identifient l'unité, le minimum et le maximum des valeurs du MDD.

Finalement, deux tables additionnelles permettraient la gestion des légendes et des couleurs indexées pour les MDD nominaux :

```
couleurs_indexées(oid, #composante, rouge, vert, bleu)
légende(oid, #composante, valeur, description)
```

1. Une énumération est un type particulier de données qui liste l'ensemble des valeurs que peut prendre un attribut

L'OID et le numéro de la composante ont le même rôle que précédemment. Pour la table des couleurs indexées, les attributs `rouge`, `vert` et `bleu` contiennent les valeurs respectives des couleurs du même nom. Pour la table de la légende, l'attribut `valeur` permet d'identifier à quelle valeur de cellule la description se rapporte. Les tables sont reprises à la figure 6.2.

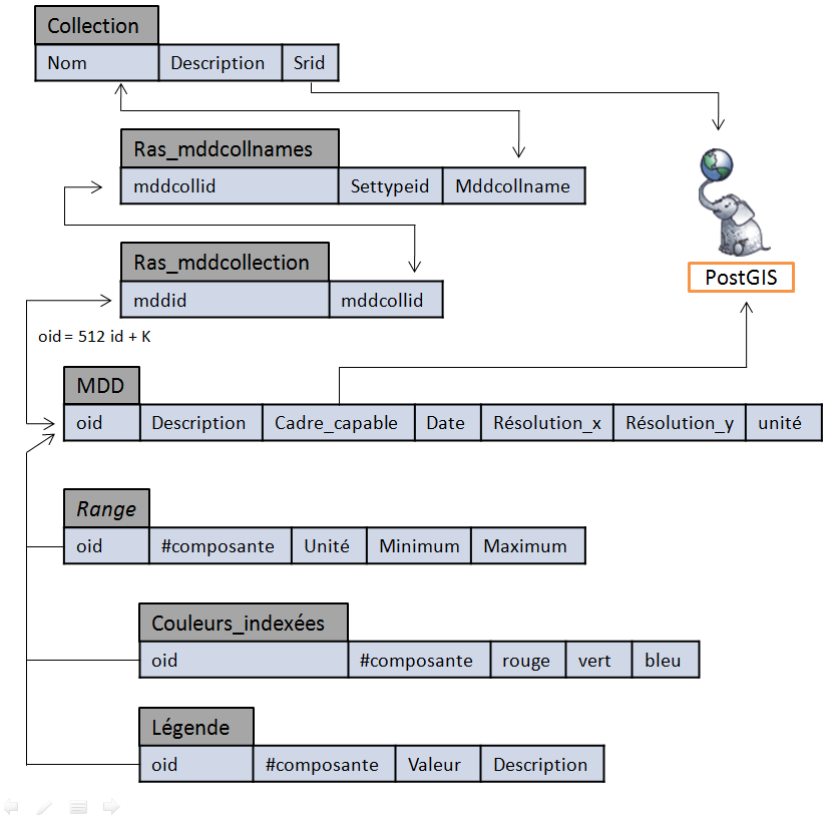


FIGURE 6.2 – Les tables permettant l'enrichissement de Rasdaman

Comme dit précédemment, il s'agit là d'une manière de faire, reposant sur les hypothèses mentionnées plus haut. Si le serveur doit gérer les MDD en trois dimensions, le modèle logique pourrait s'inspirer plus de celui de PetaScope, par exemple, en créant une table pour gérer les dimensions.

6.2.3 Avantages et inconvénients

L'architecture présentée permettrait de pallier à de nombreux problèmes de Rasdaman. Les systèmes de références et autres métadonnées sont maintenant pris en charge. Il est possible d'effectuer des projections cartographiques. Les requêtes sur les métadonnées fonctionnent. Il est possible de poser des requêtes se basant sur les coordonnées géographiques grâce au cadre capable. Le mosaïquage est facilité. Et il est possible de construire un catalogue de données grâces aux descriptions. On peut également créer des fonctions pour Rasdaman grâce à la nouvelle couche. Certaines requêtes, comme l'amélioration de contraste,

s'effectuent beaucoup plus rapidement grâce aux maximum et minimum. Finalement, d'autres requêtes, telles que le calcul de l'histogramme, peuvent être dédiées à GDAL qui les effectuent plus rapidement.

L'ajout de ces fonctionnalités permet d'enrichir Rasdaman mais un point crucial ne change pas : il s'agit toujours d'un système hybride. Ainsi, les requêtes doivent procéder en deux temps : d'abord extraire les métadonnées et puis s'occuper des MDD. Néanmoins, les métadonnées accessibles sont bien plus riches.

Chapitre 7

Conclusion

Il est maintenant temps de conclure ce mémoire. L'objectif était de faire le point sur Rasdaman et plus précisément d'analyser ses capacités comme serveur de données géographiques maillées dans le cadre du projet CARE GITAN qui vise à rassembler toutes les informations maillées de l'ULg..

Pour ce faire, nous avons procédé en plusieurs étapes. Après un état de l'art sur la normalisation des informations géographiques maillées et les systèmes permettant de les gérer (chapitre 2), nous avons posé l'objectif du mémoire. Nous avons ensuite reformulé cet objectif sous la forme suivante :

> « Rasdaman, à lui seul, est capable de faire office de serveur de données géographiques maillées, notamment dans le cadre du projet CARE GITAN de l'ULg ».

Suite à cela, nous avons dressé la liste des critères nous permettant de statuer sur l'hypothèse (chapitre 3). Ces critères concernaient les fonctionnalités du serveur de données mais aussi les données et, les traitements et requêtes qu'il pouvait gérer.

Ces critères nous ont accompagnés dans les trois parties suivantes pour les confronter à nos découvertes. Le quatrième chapitre présentait la synthèse des informations sur Rasdaman. Nous y avons exploré les composants de Rasdaman ainsi que son architecture. Nous avons appris que Rasdaman se positionnait comme un *middleware* entre un système de gestion de base de données (SGBD), ici PostgreSQL, et le client. Nous avons également découvert que Rasdaman s'inscrivait dans le paradigme de la gestion hybride des données géographiques puisqu'il ne gérait pas lui-même les métadonnées.

Nous avons alors décrit l'approche conceptuelle de Rasdaman pour gérer les *rasters*. Ceux-ci sont vus comme des tableaux dans le sens informatique et appelés *Multidimensional Discrete Data* ou MDD. Ils peuvent avoir un nombre infini de dimensions et une extension illimitée. De plus, le type de cellules peut être un composé complexe des types primitifs.

Nous nous sommes ensuite intéressés à la gestion interne des informations. Nous avons vu que celle-ci était compliquée pour offrir la souplesse du modèle conceptuel. Nous avons également passé en revue les langages de définition et de manipulation des données afin de pouvoir les exploiter par la suite.

Finalement, après un mot sur l'extension PetaScope de Rasdaman, nous

avons confronté nos découvertes aux critères de validation. Il en est ressorti que Rasdaman semblait capable de gérer un grand nombre de données et offrait un riche langage de requêtes. En revanche, Rasdaman, de par son architecture hybride, ne savait pas gérer les métadonnées. C'est sur cette conclusion que se clôturait le quatrième chapitre.

Le cinquième chapitre visait l'élaboration d'un prototype. Après quelques mots sur l'installation de Rasdaman et l'environnement dans lequel le prototype allait être conçu, nous avons discuté du jeu de données. Celui-ci reprenait quatre ensembles de données. Le premier reprenait deux images Landsat TM expressément prévues pour le mosaïquage mais qui nous ont aussi servi pour tester d'autres traitements. Le second reprenait un ensemble d'images SPOT visant à tester les flexibilités de Rasdaman en matière de gestion de données. Le troisième était constitué d'un seul élément : un modèle numérique de surface. Celui-ci a permis de tester les capacités de stockage et de traitements de Rasdaman pour les *rasters* représentant des champs géographiques. Le dernier ensemble reprenait des images Quickbird. Il a servi à tester le stockage des informations codées sur 10 bits et pour discuter l'intégration de données à résolutions spatiales différentes. On a pu voir que celle-ci n'était pas possible sous forme d'un seul MDD sans devoir modifier les images de départ.

Avec ce jeu de données, nous avons pu créer le modèle conceptuel du prototype. Il reposait sur sept collections. Les images Landsat et Quickbird ainsi que le modèle numérique de surface avaient chacun leur collection propre. Les images SPOT avaient quatre collections pour elles. Une première collection reprenait chaque image sous forme de MDD individuels. Une seconde collection testait l'intégration d'une composition colorée dans la base de données. La troisième collection reprenait toutes les images SPOT sous forme d'un seul MDD composé à deux dimensions. Finalement, la quatrième collection reprenait toutes les images sous forme d'un MDD primitif à trois dimensions : deux dimensions spatiales et une dimension spectrale.

Le modèle conceptuel créé, il ne restait plus qu'à charger les données. Le chargement s'est bien passé et très rapidement, à l'exception de deux collections se basant sur les images SPOT : la collection contenant le MDD composé et la collection contenant le MDD à trois dimensions. Le premier de ces MDD a mis en évidence une lacune du langage pour créer dynamiquement des MDD à type de cellule composé à partir d'autres MDD. Le second échec, pour le MDD à trois dimensions, serait dû à une erreur lors de la mise à jour. En outre, nous avons aussi pu vérifier que l'insertion/extraction n'était pas neutre vis-à-vis des métadonnées.

Une fois les données chargées, nous sommes passés aux traitements et aux requêtes. Pour chaque traitement, il a fallu le formuler dans le langage de requête, valider le résultat et mesurer les performances. La validation a pu se faire grâce à des logiciels tiers tels que Idrisi de Clark Lab et GDAL (*Geospatial Data Abstract Library*). Les mesures de performances ont été réalisées via la commande `time` de linux. Nous avons testé l'extraction des données, les changements de types de cellules, l'extraction de statistiques, les opérations binaires, le masquage et la reclassification, le mosaïquage, les changements d'échelles, les histogrammes, l'amélioration de contraste, les fenêtre de convolutions et des requêtes sur les MDD. Les traitements arrivés à terme ont été validés. Une partie des traitements sont très performants mais pas tous. Ainsi, les histogrammes, l'amélioration de contraste et le mosaïquage font parties des points faibles de

Rasdaman.

Dans le sixième chapitre, nous sommes revenus sur l'hypothèse. Nous avons fait le bilan des critères de validation et avons malheureusement dû rejeter l'hypothèse. La raison principale étant le manque de gestion des métadonnées en général et des métadonnées géographiques en particulier. Nous avons alors discuté de l'enrichissement de Rasdaman grâce à PostGIS (une extension de PostgreSQL pour les données vectorielles) et GDAL par la construction d'une couche supplémentaire. Nous avons vu que ces apports pouvaient combler la plupart des manques de Rasdaman : GDAL offre les capacités d'interaction avec les métadonnées et PostGIS offre le support des systèmes de coordonnées et la gestion du cadre capable des *rasters*. Le seul bémol restait l'architecture hybride du serveur de données obligeant à effectuer les requêtes en deux temps.

Voilà qui fait le point sur la version 8.0. de Rasdaman. En tant que serveur de données **géographiques** maillées, Rasdaman peut encore s'améliorer. Nous pouvons garder bon espoir qu'il le fera. Démarrant comme serveur de données maillées indépendant de tout domaine, Rasdaman a multiplié ses incursions dans le domaine géographique : PetaScope en témoigne mais aussi les dernières nouvelles sur le site de Rasdaman (**?**). Rasdaman a également fait l'objet d'un pilote pour GDAL ainsi que d'une intégration dans le DVD de l'OSGEO (la fondation géospatiale *open source* : www.osgeo.org). Ce DVD reprend un grand nombre de projets *open source* dans le domaine de la géomatique. Enfin, deux projets importants sont en conception : une intégration avec MapServer (http ://mapserver.org) et une intégration avec PostGIS. Peu d'informations sont disponibles sur ces projets mais il faut espérer qu'ils amèneront à combler les manques de Rasdaman, notamment dans la gestion des métadonnées spatiales. Nous avons déjà vu que PetaScope évoluait dans ce sens. C'est aussi la pression exercée par l'intégration avec MapServer comme Mr. Owonibi, responsable de ce projet, le confie dans le groupe de discussion :

> [...]*we need to improve on 2 things namely : geo-coordinates handling, and bounding-box-aware automatic query composition.*

On peut donc espérer que d'ici quelques versions, Rasdaman comblera les manques qu'on a pu soulever au cours de ce mémoire.

Bibliographie

ADAM N., GANGOPADHYAY A. *Database Issues in Geographic Information Systems*. Kluwer Academic, 1e edition, 1997.

BECLA J. et LIM K.T. Report from the SciDB workshop. *Data Science Journal*, 7(0) : pages 88-95, 2008.

BAUMANN P. Installation and administration guide - rasdaman version 8.0, 2009a.

BAUMANN P. Query language guide - rasdaman version 8.0, 2009b.

BAUMANN P. Beyond rasters : introducing the new OGC web coverage service 2.0. In Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 320-329. ACM, 2010a.

BAUMANN P. The OGC web coverage processing service (WCPS) standard. *Geoinformatica*, pages 1-33, 2010b.

BAUMANN P. Putting pixels in place : A storage layout language for scientic data. *2010 IEEE International Conference on Data Mining Workshops*. IEEE. 2010c.

BAUMANN P. et AIORD A. Petascope : an open-source implementation of the ogc wcs geo service standards suite. *Scientic and Statistical Database Management*, pages 160-168. Springer, 2010.

BAUMANN P. et WIDMANN N. Efficient execution of operations in a dbms for multidimensional arrays. *ssdbm*, page 1-55. IEEE Computer Society, 1998.

COPPOCK J.T., RHIND D.W. The history of GIS. *Geographical information systems : Principles and applications*, 1 :pages 21-43, 1991.

DONNAY J.P. Formalisation des informations géographiques en mode maillé. *Revue Internationale de Géomatique*, 15(4), 2005.

DONNAY J.P. Analyse spatiale et S.I.G. Note de cours. Université de Liège, Faculté des sciences. Indédit, 2008.

DONNAY J.P. et CORNET Y. Télédétection. Note de cours. Université de Liège, Faculté des sciences, Inédit, 2008.

ESRI. Raster data in ArcSDE 9.1, 2005.

FURATDO *et al.* Object-oriented design of a database engine for multidimensional discrete data. *Proc. of the OOIS*, volume 97. Citeseer, 1997.

GARDARIN G. *Bases de données*. Eyrolles, 5e edition, 2003.

GERTZ M. *et al.* A data and query model for streaming geospatial image data. *Current Trends in Database Technology EDBT*, 2006.

ISAAKS E. et SRIVASTAVA R. *An Introduction To Applied Geostatistics*. Oxford University Press, Oxford, 1989.

ISO/TC 211. ISO 19101 : Geographic Information - Reference Model, 2002.

ISO/TC 211. ISO 19107 : Geographic Information - Spatial Schema, 2003.

KUROSE J. et ROSS K. *Computer Networking : A Top-Down Approch*. Addison Wesley, New York, 5 edition, 2009.

LIBKKIN L., MACHLIN R. et WONG L. A query language for multidimensional arrays : Design, implementation, and optimization techniques. *ACM SIGMOD Record*, volume 25, pages 228-239. ACM, 1996.

OGC. OpenGis abstract specication topic 6 : Schema for coverage geometry and functions, version 4.0, 2000a.

OGC. OpenGIS abstract specication topic 15 : Image exploitation services, version 6.0, 2000b.

OGC. OpenGIS abstract specication topic 16 : Image coordinate transformation services, version 4, 2000c.

OGC. OpenGIS abstract specication topic 7 : The earth imagery case, version 5, 2004.

OGC. OpenGis abstract specication topic 6 : Schema for coverage geometry and functions, version 7.0, 2006a.

OGC. OpenGIS web map service (WMS) implementa- tion specication, 2006b.

OGC. OpenGIS geography markup language (GML) encoding standard, 2007.

OGC. Web coverage service (WCS) implementation standard, 2008.

OGC. OpenGIS web coverage processing service (WCPS) language interface standard, 2009.

OGC. OGC WCS 2.0 interface standard - core, 2010a.

OGC. OGC web services common standard, 2010b.

OGC. OpenGIS web feature service 2.0 interface stan- dard, 2010c.

ORACLE. Oracle spatial GeoRaster, 10g release 1 (10.1), 2003.

ORACLE. Oracle spatial 11g - GeoRaster, 2007.

POSTGRESQL. PostgreSQL 8.3.15 Documentation. 2008.

RASDAMAN (www.rasdaman.org), consulté le 16 aout 2011.

REINER B. *et al.* Hierarchical storage support and management for large-scale multidimensional array database management systems. *Database and Expert Systems Applications*, pages 129-151. Springer, 2002.

REINER B. et HAHN K. Optimized management of large-scale data sets stored on tertiary storage systems. Distributed Systems Online, IEEE, 5(5), 2004.

RITSCH R. *et al.* Geo/environmental and medical data management in the RasDaMan system. *Proceedings of the international conference on very large data bases*, IEEE. Pages 548-552. 1997.

SELLIS T. *et al.* The r+-tree : A dynamic in- dex for multi-dimensional objects. *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 507-518. Citeseer, 1987.

STONEBRAKER M. Sequoia 2000 : A reflection on the first three years. *Computational Science and Engineering*, IEEE. Pages 63-72, 1994.

TOMLIN C.D. Cartographic modelling. *Geographical Information Systems*, 1 : pages 361-374, 1991.

WORBOYS M. et DUCKHAM M. *GIS : A Computing Perspective*, Second Edition. CRC Press, 2e édition, 2004.

Annexe A - Les données

Dans cette première annexe, sont reprises les données qui ont servi à l'élaboration du prototype. Les métadonnées ont été extraites par GDAL.

Landsat TM

Les deux images Landsat TM sont reprises à la figure 7.1. La première a les métadonnées suivantes :

```
Driver: GTiff/GeoTIFF
Files: p180r031.tif
Size is 1024, 600
Coordinate System is:
PROJCS["WGS 84 / UTM zone 35N",
 GEOGCS["WGS 84",
  DATUM["WGS_1984",
   SPHEROID["WGS 84",6378137,298.2572235629972,
    AUTHORITY["EPSG","7030"]],
   AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]],
 PROJECTION["Transverse_Mercator"],
 PARAMETER["latitude_of_origin",0],
 PARAMETER["central_meridian",27],
 PARAMETER["scale_factor",0.9996],
 PARAMETER["false_easting",500000],
 PARAMETER["false_northing",0],
 UNIT["metre",1,
  AUTHORITY["EPSG","9001"]],
 AUTHORITY["EPSG","32635"]]
Origin = (665232.750,4558218.75)
Pixel Size = (14.25,-14.25)
Metadata:   AREA_OR_POINT=Area
TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
```

```
@(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $ $
Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (  665232.750, 4558218.750)
 ( 28d58'9.67"E, 41d 9'30.51"N)
Lower Left  (  665232.750, 4549668.750)
 ( 28d58'1.39"E, 41d 4'53.40"N)
Upper Right (  679824.750, 4558218.750)
 ( 29d 8'35.37"E, 41d 9'19.33"N)
Lower Right (  679824.750, 4549668.750)
 ( 29d 8'26.36"E, 41d 4'42.26"N)
Center      (  672528.750, 4553943.750)
 ( 29d 3'18.21"E, 41d 7'6.49"N)
Band 1 Block=1024x8 Type=Byte, ColorInterp=Gray
 Min=1.000 Max=255.000
 Minimum=1, Maximum=255, Mean=64.187, StdDev=19.714
 Metadata:
  STATISTICS_MINIMUM=1
  STATISTICS_MAXIMUM=255
  STATISTICS_MEAN=64.187283528646
  STATISTICS_MEDIAN=67
  STATISTICS_MODE=68
  STATISTICS_STDDEV=19.714329003747
  STATISTICS_HISTONUMBINS=256
  STATISTICS_HISTOMIN=0
  STATISTICS_HISTOMAX=255
```

La seconde :

```
Driver: GTiff/GeoTIFF
Files: p180r032.tif
Size is 1024, 600
Coordinate System is:
PROJCS["WGS 84 / UTM zone 35N",
 GEOGCS["WGS 84",
  DATUM["WGS_1984",
   SPHEROID["WGS 84",6378137,298.2572235629972,
    AUTHORITY["EPSG","7030"]],
   AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]],
 PROJECTION["Transverse_Mercator"],
 PARAMETER["latitude_of_origin",0],
 PARAMETER["central_meridian",27],
 PARAMETER["scale_factor",0.9996],
```

```
 PARAMETER["false_easting",500000],
 PARAMETER["false_northing",0],
 UNIT["metre",1,
  AUTHORITY["EPSG","9001"]],
 AUTHORITY["EPSG","32635"]]
Origin = (665232.75,4558218.75)
Pixel Size = (14.25,-14.25)
Metadata:   AREA_OR_POINT=Area
TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
@(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $ $
Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (  665232.750, 4551093.750)
 ( 28d58'2.77"E, 41d 5'39.59"N)
Lower Left  (  665232.750, 4549668.750)
 ( 28d58'1.39"E, 41d 4'53.40"N)
Upper Right (  679824.750, 4551093.750)
 ( 29d 8'27.86"E, 41d 5'28.44"N)
Lower Right (  679824.750, 4542543.750)
 ( 29d 8'18.88"E, 41d 0'51.36"N)
Center      (  672528.750, 4546818.750)
 ( 29d 3'11.01"E, 41d 3'15.58"N)
Band 1 Block=1024x8 Type=Byte, ColorInterp=Gray
 Min=1.000 Max=255.000
 Minimum=1, Maximum=255, Mean=64.142, StdDev=19.709
 Metadata:
 STATISTICS_MINIMUM=1
 STATISTICS_MAXIMUM=255
 STATISTICS_MEAN=64.142342122396
 STATISTICS_MEDIAN=66
 STATISTICS_MODE=67
 STATISTICS_STDDEV=19.709131920356
 STATISTICS_HISTONUMBINS=256
 STATISTICS_HISTOMIN=0
 STATISTICS_HISTOMAX=255
```

SPOT

Les images SPOT sont reprises dans l'ordre aux figures 7.2, 7.3, 7.4et 7.5. La première image a pour métadonnées :

```
Driver: GTiff/GeoTIFF
```

```
Files: xs1.tif
Size is 1024, 600
Coordinate System is:
PROJCS["WGS 84 / UTM zone 35N",
 GEOGCS["WGS 84",
  DATUM["WGS_1984",
   SPHEROID["WGS 84",6378137,298.2572235629972,
    AUTHORITY["EPSG","7030"]],
   AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]],
 PROJECTION["Transverse_Mercator"],
 PARAMETER["latitude_of_origin",0],
 PARAMETER["central_meridian",27],
 PARAMETER["scale_factor",0.9996],
 PARAMETER["false_easting",500000],
 PARAMETER["false_northing",0],
 UNIT["metre",1,
  AUTHORITY["EPSG","9001"]],
 AUTHORITY["EPSG","32635"]]
Origin = (656960.000000000000000,4546070.000000000000000)
Pixel Size = (10.000000000000000,-10.000000000000000)
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
@(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 656960.000, 4546070.000)
 ( 28d52'3.73"E, 41d 3'2.66"N)
Lower Left ( 656960.000, 4540070.000)
 ( 28d51'58.24"E, 40d59'48.19"N)
Upper Right ( 667200.000, 4546070.000)
 ( 28d59'22.14"E, 41d 2'55.32"N)
Lower Right ( 667200.000, 4540070.000)
 ( 28d59'16.29"E, 40d59'40.86"N)
Center ( 662080.000, 4543070.000)
 ( 28d55'40.10"E, 41d 1'21.82"N)
Band 1 Block=1024x8 Type=Byte, ColorInterp=Gray
 Min=14.000 Max=194.000
 Minimum=14, Maximum=194, Mean=63.4, StdDev=20.374
 Metadata:
 STATISTICS_MINIMUM=14
```

```
 STATISTICS_MAXIMUM=194
 STATISTICS_MEAN=63.400133463542
 STATISTICS_MEDIAN=65
 STATISTICS_MODE=66
 STATISTICS_STDDEV=20.373791961586
```

La seconde a pour métadonnées :

```
Driver: GTiff/GeoTIFF
Files: xs2.tif
Size is 1024, 600
Coordinate System is:
PROJCS["WGS 84 / UTM zone 35N",
 GEOGCS["WGS 84",
  DATUM["WGS_1984",
   SPHEROID["WGS 84",6378137,298.2572235629972,
    AUTHORITY["EPSG","7030"]],
   AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]],
 PROJECTION["Transverse_Mercator"],
 PARAMETER["latitude_of_origin",0],
 PARAMETER["central_meridian",27],
 PARAMETER["scale_factor",0.9996],
 PARAMETER["false_easting",500000],
 PARAMETER["false_northing",0],
 UNIT["metre",1,
  AUTHORITY["EPSG","9001"]],
 AUTHORITY["EPSG","32635"]]
Origin = (656960.000000000000000,4546070.000000000000000)
Pixel Size = (10.000000000000000,-10.000000000000000)
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
@(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 656960.000, 4546070.000)
 ( 28d52'3.73"E, 41d 3'2.66"N)
Lower Left ( 656960.000, 4540070.000)
 ( 28d51'58.24"E, 40d59'48.19"N)
Upper Right ( 667200.000, 4546070.000)
 ( 28d59'22.14"E, 41d 2'55.32"N)
```

```
Lower Right ( 667200.000, 4540070.000)
 ( 28d59'16.29"E, 40d59'40.86"N)
Center ( 662080.000, 4543070.000)
 ( 28d55'40.10"E, 41d 1'21.82"N)
Band 1 Block=1024x8 Type=Byte, ColorInterp=Gray
 Min=53.000 Max=255.000
 Minimum=53, Maximum=255, Mean=123.759, StdDev=33.705
 Metadata:
  STATISTICS_MINIMUM=53
  STATISTICS_MAXIMUM=255
  STATISTICS_MEAN=123.75912597656
  STATISTICS_MEDIAN=125
  STATISTICS_MODE=129
  STATISTICS_STDDEV=33.704607817096
```

Les métadonnées de la troisième sont :

```
Driver: GTiff/GeoTIFF
Files: xs2.tif
Size is 1024, 600
Coordinate System is:
PROJCS["WGS 84 / UTM zone 35N",
 GEOGCS["WGS 84",
  DATUM["WGS_1984",
   SPHEROID["WGS 84",6378137,298.2572235629972,
    AUTHORITY["EPSG","7030"]],
   AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]],
 PROJECTION["Transverse_Mercator"],
 PARAMETER["latitude_of_origin",0],
 PARAMETER["central_meridian",27],
 PARAMETER["scale_factor",0.9996],
 PARAMETER["false_easting",500000],
 PARAMETER["false_northing",0],
 UNIT["metre",1,
  AUTHORITY["EPSG","9001"]],
 AUTHORITY["EPSG","32635"]]
Origin = (656960.000000000000000,4546070.000000000000000)
Pixel Size = (10.000000000000000,-10.000000000000000)
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
@(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
```

```
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 656960.000, 4546070.000)
 ( 28d52'3.73"E, 41d 3'2.66"N)
Lower Left ( 656960.000, 4540070.000)
 ( 28d51'58.24"E, 40d59'48.19"N)
Upper Right ( 667200.000, 4546070.000)
 ( 28d59'22.14"E, 41d 2'55.32"N)
Lower Right ( 667200.000, 4540070.000)
 ( 28d59'16.29"E, 40d59'40.86"N)
Center ( 662080.000, 4543070.000)
 ( 28d55'40.10"E, 41d 1'21.82"N)
Band 1 Block=1024x8 Type=Byte, ColorInterp=Gray
 Min=73.000 Max=255.000
 Minimum=73, Maximum=255, Mean=115.167, StdDev=21.752
 Metadata:
  STATISTICS_MINIMUM=73
  STATISTICS_MAXIMUM=255
  STATISTICS_MEAN=115.16666015625
  STATISTICS_MEDIAN=114
  STATISTICS_MODE=114
  STATISTICS_STDDEV=21.752204069396
```

Enfin, la quatrième image a pour métadonnées :

```
Driver: GTiff/GeoTIFF
Files: xs2.tif
Size is 1024, 600
Coordinate System is:
PROJCS["WGS 84 / UTM zone 35N",
 GEOGCS["WGS 84",
  DATUM["WGS_1984",
   SPHEROID["WGS 84",6378137,298.2572235629972,
    AUTHORITY["EPSG","7030"]],
   AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]],
 PROJECTION["Transverse_Mercator"],
 PARAMETER["latitude_of_origin",0],
 PARAMETER["central_meridian",27],
 PARAMETER["scale_factor",0.9996],
 PARAMETER["false_easting",500000],
 PARAMETER["false_northing",0],
 UNIT["metre",1,
  AUTHORITY["EPSG","9001"]],
 AUTHORITY["EPSG","32635"]]
Origin = (656960.000000000000000,4546070.000000000000000)
Pixel Size = (10.000000000000000,-10.000000000000000)
```

```
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
@(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 656960.000, 4546070.000)
 ( 28d52'3.73"E, 41d 3'2.66"N)
Lower Left ( 656960.000, 4540070.000)
 ( 28d51'58.24"E, 40d59'48.19"N)
Upper Right ( 667200.000, 4546070.000)
 ( 28d59'22.14"E, 41d 2'55.32"N)
Lower Right ( 667200.000, 4540070.000)
 ( 28d59'16.29"E, 40d59'40.86"N)
Center ( 662080.000, 4543070.000)
 ( 28d55'40.10"E, 41d 1'21.82"N)
Band 1 Block=1024x8 Type=Byte, ColorInterp=Gray
  Min=10.000 Max=244.000
  Minimum=10, Maximum=244, Mean=93.337, StdDev=32.32
  Metadata:
   STATISTICS_MINIMUM=10
   STATISTICS_MAXIMUM=244
   STATISTICS_MEAN=93.337356770833
   STATISTICS_MEDIAN=99
   STATISTICS_MODE=107
   STATISTICS_STDDEV=32.320351691332
```

Quickbird

Les images Quickbird sont reprises dans l'ordre aux figures 7.6 à 7.9. La figure 7.10 reprend l'image panchromatique.

Les métadonnées de la première image sont :

```
Driver: GTiff/GeoTIFF
Files: qb-xs1.tif
Size is 256, 150
Coordinate System is:
GEOGCS["WGS 84",
 DATUM["WGS_1984",
  SPHEROID["WGS 84",6378137,298.2572235629972,
   AUTHORITY["EPSG","7030"]],
  AUTHORITY["EPSG","6326"]],
```

```
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]]
GeoTransform =
5.57009210107991, 3.55921393565381e-05,
 -6.885463444814198e-08
50.64096943434568, -3.707014738602087e-07,
 -2.334078234702523e-05
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
 @(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 5.5700921, 50.6409694)
 ( 5d34'12.33"E, 50d38'27.49"N)
Lower Left ( 5.5700818, 50.6374683)
 ( 5d34'12.29"E, 50d38'14.89"N)
Upper Right ( 5.5792037, 50.6408745)
 ( 5d34'45.13"E, 50d38'27.15"N)
Lower Right ( 5.5791934, 50.6373734)
 ( 5d34'45.10"E, 50d38'14.54"N)
Center ( 5.5746427, 50.6391714) ( 5d34'28.71"E, 50d38'21.02"N)
Band 1 Block=256x16 Type=UInt16, ColorInterp=Gray
 Min=127.000 Max=875.000
 Minimum=127, Maximum=875, Mean=168.496, StdDev=37.116
 Metadata:
  STATISTICS_MINIMUM=127
  STATISTICS_MAXIMUM=875
  STATISTICS_MEAN=168.49614583333
  STATISTICS_MEDIAN=156.21875
  STATISTICS_MODE=147.453125
  STATISTICS_STDDEV=37.115939909689
```

La seconde image a pour métadonnées :

```
Driver: GTiff/GeoTIFF
Files: qb-xs2.tif
Size is 256, 150
Coordinate System is:
GEOGCS["WGS 84",
 DATUM["WGS_1984",
  SPHEROID["WGS 84",6378137,298.2572235629972,
   AUTHORITY["EPSG","7030"]],
```

```
  AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]]
GeoTransform =
5.57009210107991, 3.55921393565381e-05,
 -6.885463444814198e-08
50.64096943434568, -3.707014738602087e-07,
 -2.334078234702523e-05
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
 @(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 5.5700921, 50.6409694)
 ( 5d34'12.33"E, 50d38'27.49"N)
Lower Left ( 5.5700818, 50.6374683)
 ( 5d34'12.29"E, 50d38'14.89"N)
Upper Right ( 5.5792037, 50.6408745)
 ( 5d34'45.13"E, 50d38'27.15"N)
Lower Right ( 5.5791934, 50.6373734)
 ( 5d34'45.10"E, 50d38'14.54"N)
Center ( 5.5746427, 50.6391714)
 ( 5d34'28.71"E, 50d38'21.02"N)
Band 1 Block=256x16 Type=UInt16, ColorInterp=Gray
  Min=137.000 Max=1505.000
  Minimum=137, Maximum=1505, Mean=212.511, StdDev=69.993
  Metadata:
   STATISTICS_MINIMUM=137
   STATISTICS_MAXIMUM=1505
   STATISTICS_MEAN=212.51052083333
   STATISTICS_MEDIAN=190.4375
   STATISTICS_MODE=174.40625
   STATISTICS_STDDEV=69.993195502052
```

La troisième image a pour métadonnées :

```
Driver: GTiff/GeoTIFF
Files: qb-xs3.tif
Size is 256, 150
Coordinate System is:
GEOGCS["WGS 84",
 DATUM["WGS_1984",
```

```
  SPHEROID["WGS 84",6378137,298.2572235629972,
   AUTHORITY["EPSG","7030"]],
  AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]]
GeoTransform =
5.57009210107991, 3.55921393565381e-05,
 -6.885463444814198e-08
50.64096943434568, -3.707014738602087e-07,
 -2.334078234702523e-05
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
 @(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 5.5700921, 50.6409694)
 ( 5d34'12.33"E, 50d38'27.49"N)
Lower Left ( 5.5700818, 50.6374683)
 ( 5d34'12.29"E, 50d38'14.89"N)
Upper Right ( 5.5792037, 50.6408745)
 ( 5d34'45.13"E, 50d38'27.15"N)
Lower Right ( 5.5791934, 50.6373734)
 ( 5d34'45.10"E, 50d38'14.54"N)
Center ( 5.5746427, 50.6391714)
 ( 5d34'28.71"E, 50d38'21.02"N)
Band 1 Block=256x16 Type=UInt16, ColorInterp=Gray
   Min=57.000 Max=1308.000
   Minimum=57, Maximum=1308, Mean=124.503, StdDev=63.361
   Metadata:
    STATISTICS_MINIMUM=57
    STATISTICS_MAXIMUM=1308
    STATISTICS_MEAN=124.503203125
    STATISTICS_MEDIAN=100.98046875
    STATISTICS_MODE=81.43359375
    STATISTICS_STDDEV=63.36124938225
```

La quatrième image a pour métadonnées :

```
Driver: GTiff/GeoTIFF
Files: qb-xs4.tif
Size is 256, 150
Coordinate System is:
```

```
GEOGCS["WGS 84",
 DATUM["WGS_1984",
  SPHEROID["WGS 84",6378137,298.2572235629972,
   AUTHORITY["EPSG","7030"]],
  AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]]
GeoTransform =
5.57009210107991, 3.55921393565381e-05,
 -6.885463444814198e-08
50.64096943434568, -3.707014738602087e-07,
 -2.334078234702523e-05
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
 @(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 5.5700921, 50.6409694)
 ( 5d34'12.33"E, 50d38'27.49"N)
Lower Left ( 5.5700818, 50.6374683)
 ( 5d34'12.29"E, 50d38'14.89"N)
Upper Right ( 5.5792037, 50.6408745)
 ( 5d34'45.13"E, 50d38'27.15"N)
Lower Right ( 5.5791934, 50.6373734)
 ( 5d34'45.10"E, 50d38'14.54"N)
Center ( 5.5746427, 50.6391714)
 ( 5d34'28.71"E, 50d38'21.02"N)
Band 1 Block=256x16 Type=UInt16, ColorInterp=Gray
 Min=36.000 Max=1368.000
 Minimum=36, Maximum=1368, Mean=129.919, StdDev=89.283
 Metadata:
  STATISTICS_MINIMUM=36
  STATISTICS_MAXIMUM=1368
  STATISTICS_MEAN=129.91932291667
  STATISTICS_MEDIAN=108.84375
  STATISTICS_MODE=46.40625
  STATISTICS_STDDEV=89.282509460387
```

Enfin, l'image pancrhomatique a pour métadonnées :

```
Driver: GTiff/GeoTIFF
Files: qb-xs4.tif
```

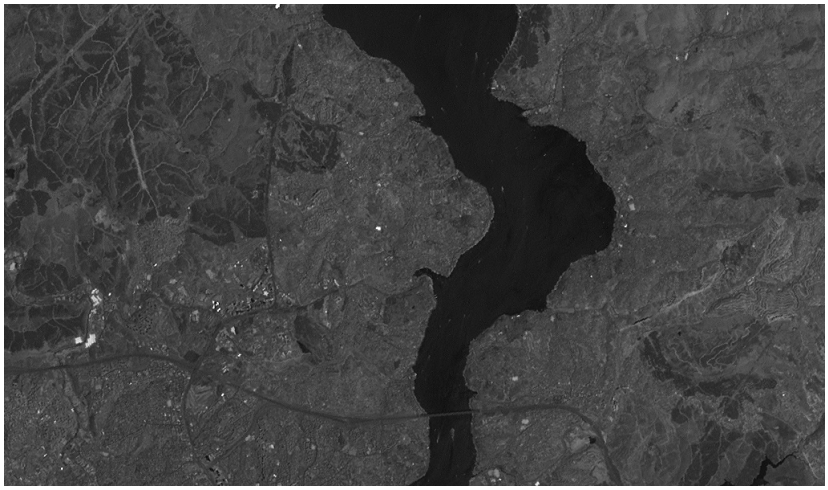
```
Size is 1024, 600
Coordinate System is:
GEOGCS["WGS 84",
 DATUM["WGS_1984",
  SPHEROID["WGS 84",6378137,298.2572235629972,
   AUTHORITY["EPSG","7030"]],
  AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]]
GeoTransform =
  5.570071529485361, 8.910738868205557e-06,
    -1.583492748969477e-08
  50.64098026539211, -9.528222932769122e-08,
    -5.831156853423003e-06
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved
 @(#)$RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (   5.5700715,  50.6409803)
 (  5d34'12.26"E, 50d38'27.53"N)
Lower Left  (   5.5700620,  50.6374816)
 (  5d34'12.22"E, 50d38'14.93"N)
Upper Right (   5.5791961,  50.6408827)
 (  5d34'45.11"E, 50d38'27.18"N)
Lower Right (   5.5791866,  50.6373840)
 (  5d34'45.07"E, 50d38'14.58"N)
Center      (   5.5746291,  50.6391821)
 (  5d34'28.66"E, 50d38'21.06"N)
Band 1 Block=256x16 Type=UInt16, ColorInterp=Gray
  Min=97.000 Max=1971.000
  Minimum=97, Maximum=1971, Mean=212.041, StdDev=116.273
  Metadata:
   STATISTICS_MINIMUM=97
   STATISTICS_MAXIMUM=1971
   STATISTICS_MEAN=212.04060384115
   STATISTICS_MEDIAN=170.203125
   STATISTICS_MODE=126.28125
   STATISTICS_STDDEV=116.27275236162
```

Le modèle numérique de surface

Le modèle numérique de surface est repris à la figure 7.11. Ses métadonnées sont :

```
Driver: GTiff/GeoTIFF
Files: mns.tif
Size is 1024, 600
Coordinate System is:
PROJCS["IMAGINE GeoTIFF Support ERDAS Desktop 2011 11.0.0.203
 Projection Name = Lambert Conformal Conic
 Units = meters GeoTIFF Units = meters",
 GEOGCS["IMAGINE GeoTIFF Support ERDAS Desktop 2011 11.0.0.203
 Unable to match Ellipsoid (Datum) to a GeographicTypeGeoKey value
 Ellipsoid = International 1924 Datum = Belge 1972-1",
  DATUM["unknown",
   SPHEROID["unnamed",6378388,297.0000000284045]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433]],
 PROJECTION["Lambert_Conformal_Conic_2SP"],
 PARAMETER["standard_parallel_1",49.8333339],
 PARAMETER["standard_parallel_2",51.16666733333334],
 PARAMETER["latitude_of_origin",90.00000000000003],
 PARAMETER["central_meridian",4.367486666666667],
 PARAMETER["false_easting",150000.01256],
 PARAMETER["false_northing",5400088.4378],
 UNIT["metre",1, AUTHORITY["EPSG","9001"]]]
Origin = (239005,145495)
Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=IMAGINE TIFF Support
Copyright 1991 - 1999 by ERDAS, Inc. All Rights Reserved @(#)
 $RCSfile: etif.c $ $Revision: 1.10.1.9.1.9.2.11 $
$Date: 2004/09/15 18:42:01EDT $
 TIFFTAG_XRESOLUTION=1
 TIFFTAG_YRESOLUTION=1
 TIFFTAG_RESOLUTIONUNIT=1 (unitless)
Image Structure Metadata:
 INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 239005.000, 145495.000)
 ( 5d37'30.29"E, 50d36'50.01"N)
Lower Left ( 239005.000, 142495.000)
 ( 5d37'27.71"E, 50d35'12.93"N)
Upper Right ( 244125.000, 145495.000)
 ( 5d41'50.67"E, 50d36'47.12"N)
Lower Right ( 244125.000, 142495.000)
 ( 5d41'47.94"E, 50d35'10.05"N)
Center ( 241565.000, 143995.000)
 ( 5d39'39.15"E, 50d36'0.05"N)
```

```
Band 1 Block=512x512 Type=Int16, ColorInterp=Gray
 Min=668.000 Max=2875.000
 Minimum=668.000, Maximum=2875.000, Mean=1843.231, StdDev=563.628
 Overviews: 512x300, 256x150, 128x75, 64x37
 Metadata:
  STATISTICS_MINIMUM=668
  STATISTICS_MAXIMUM=2875
  STATISTICS_MEAN=1843.2312386068
  STATISTICS_MEDIAN=1971
  STATISTICS_MODE=2261
  STATISTICS_STDDEV=563.62829011023
```



(a) L'image Landsat nord

(b) L'image Landsat sud

FIGURE 7.1 – Les images Landsat

FIGURE 7.2 – La première image SPOT

FIGURE 7.3 – La seconde image SPOT

FIGURE 7.4 – La troisième image SPOT

FIGURE 7.5 – La quatrième image SPOT

FIGURE 7.6 – La première image Quickbird

FIGURE 7.7 – La seconde image Quickbird

FIGURE 7.8 – La troisième image Quickbird

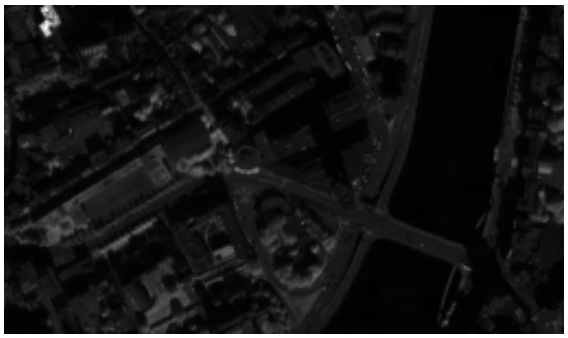


FIGURE 7.9 – La quatrième image Quickbird

FIGURE 7.10 – L'image panchromatique Quickbird

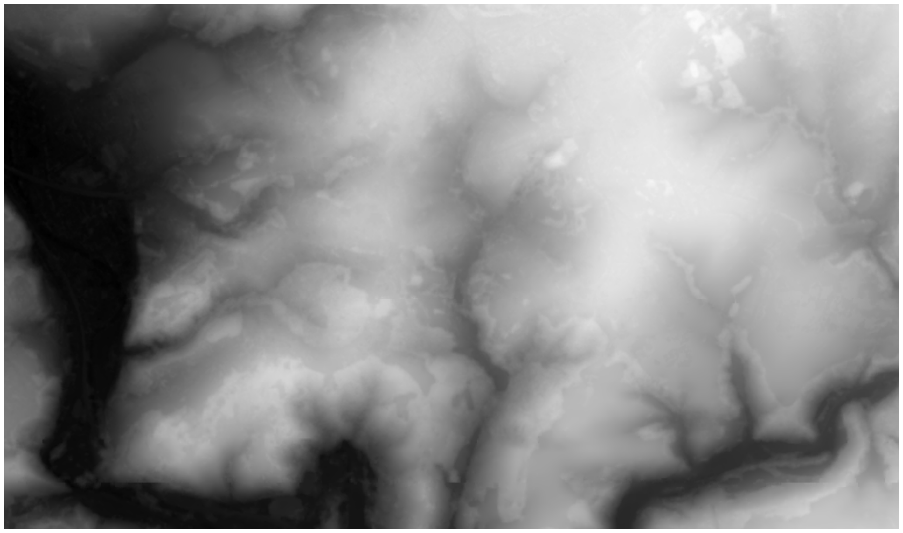


FIGURE 7.11 – Le modèle numérique de surface

Annexe B - Les traitements

Cette annexe liste les observations qui ont servi au calcul des statistiques des mesures de performances des traitements. Celles-ci sont évidemment exprimées en secondes.

Extraction des données

```
select tiff(mdd) from landsatcol as mdd
```

0.25 0.23 0.21 0.21 0.21 0.2 0.21 0.21 0.22 0.2 0.21 0.21 0.21 0.21 0.21 0.2 0.21 0.21 0.22 0.22 0.2 0.21 0.2 0.2 0.21 0.2 0.21 0.2 0.2 0.21 0.2 0.2 0.28 0.26 0.2 0.27 0.2 0.2 0.2 0.2 0.2 0.21 0.2 0.2 0.2 0.2 0.2 0.21 0.21 0.21 0.2 0.2 0.2 0.2 0.34 0.36 0.2 0.2 0.2 0.2 0.2 0.24 0.2 0.2 0.21 0.2 0.21 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.21 0.2 0.2 0.2 0.2 0.21 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.21 0.2 0.2 0.2 0.2 0.2 0.2 0.22

```
select tiff(mdd[0:500, 0:500]) from landsatcol as mdd
```

0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.18 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.19 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.13

```
select mdd[25,25] from landsatcol as mdd
```

0.08 0.07 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.08 0.08 0.08 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.08 0.08 0.06 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08

Les changements de type de cellules

```
select tiff((char)mdd) from quickcol as mdd
```

0.29 0.3 0.3 0.32 0.45 0.53 0.35 0.3 0.3 0.31 0.3 0.31 0.3 0.3 0.31 0.3 0.3 0.3 0.3
0.31 0.3 0.3 0.3 0.3 0.31 0.3 0.32 0.4 0.3 0.31 0.3 0.31 0.3 0.3 0.32 0.3 0.32 0.3
0.3 0.32 0.3 0.31 0.3 0.3 0.31 0.3 0.32 0.3 0.3 0.67 0.47 0.3 0.34 0.3 0.31 0.3 0.3
0.32 0.3 0.31 0.3 0.3 0.32 0.3 0.31 0.3 0.3 0.31 0.3 0.32 0.3 0.3 0.32 0.3 0.41 0.3
0.31 0.31 0.3 0.31 0.3 0.3 0.31 0.38 0.31 0.3 0.3 0.33 0.3 0.32 0.35 0.3 0.32 0.38
0.65 0.34 0.29 0.3 0.31 0.3

Les statistiques

Minimum, maximum et moyenne

```
select min_cells(mdd) from landsatcol as mdd
```

0.21 0.14 0.14 0.14 0.14 0.15 0.14 0.15 0.14 0.14 0.14 0.14 0.15 0.14 0.14 0.22
0.18 0.13 0.14 0.15 0.14 0.14 0.15 0.15 0.14 0.14 0.15 0.15 0.15 0.14 0.14 0.14
0.15 0.14 0.15 0.14 0.15 0.14 0.14 0.15 0.14 0.14 0.15 0.15 0.14 0.15 0.15 0.14
0.15 0.14 0.14 0.14 0.15 0.14 0.14 0.14 0.15 0.15 0.15 0.15 0.15 0.14 0.27 0.15
0.15 0.15 0.14 0.14 0.14 0.14 0.14 0.14 0.15 0.25 0.21 0.18 0.16 0.14 0.15 0.15
0.14 0.14 0.14 0.15 0.18 0.15 0.14 0.15 0.19 0.15 0.15 0.14 0.15 0.18 0.14 0.15
0.15 0.15 0.14 0.14

```
select max_cells(mdd) from landsatcol as mdd
```

0.16 0.14 0.14 0.15 0.15 0.14 0.15 0.15 0.14 0.14 0.15 0.15 0.15 0.14 0.15 0.15
0.15 0.14 0.14 0.15 0.15 0.14 0.14 0.15 0.14 0.15 0.15 0.15 0.14 0.17 0.17 0.15
0.15 0.15 0.15 0.15 0.15 0.14 0.14 0.15 0.14 0.15 0.15 0.14 0.15 0.14 0.15 0.14
0.14 0.15 0.14 0.15 0.15 0.15 0.15 0.15 0.14 0.15 0.15 0.14 0.15 0.15 0.16 0.15
0.15 0.15 0.15 0.14 0.15 0.14 0.15 0.15 0.15 0.14 0.14 0.14 0.14 0.15 0.15 0.15
0.15 0.14 0.15 0.15 0.15 0.15 0.14 0.14 0.14 0.14 0.15 0.16 0.16 0.15 0.14 0.14
0.15 0.14 0.15 0.15

```
select avg_cells(mdd) from landsatcol as mdd
```

0.18 0.15 0.15 0.15 0.15 0.15 0.15 0.16 0.16 0.15 0.15 0.16 0.15 0.16 0.16 0.16
0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.16 0.16 0.15 0.15 0.16 0.15 0.15 0.18
0.17 0.15 0.15 0.15 0.15 0.16 0.16 0.15 0.15 0.15 0.15 0.15 0.16 0.18 0.15 0.15
0.15 0.17 0.15 0.15 0.15 0.15 0.15 0.15 0.22 0.22 0.27 0.16 0.15 0.16 0.16 0.16
0.15 0.15 0.15 0.16 0.16 0.15 0.15 0.15 0.15 0.15 0.15 0.16 0.15 0.15 0.15 0.16
0.15 0.23 0.18 0.15 0.15 0.16 0.15 0.15 0.15 0.15 0.15 0.19 0.16 0.15 0.15 0.15
0.16 0.15 0.16 0.15

Écart type

```
select
    sqrt(
        condense +
        over x in sdom(mdd)
        using (((double) mdd[x] - 64.19d)
             *((double) mdd[x] - 64.19d)))
/614399d)
from landsatcol as mdd
```

10.04 10.04 10.07 9.99 10.00 10.10 10.03 10.02 10.01 10.02 10.21 10.02 10.03 10.02 10.00 10.12 10.03 10.07 10.10 9.97 10.04 10.11 10.30 10.07 9.99 10.06 10.06 10.00 10.03 9.99 10.07 10.05 10.03 10.06 10.03 10.09 10.09 10.26 10.06 10.12 10.06 10.11 10.04 10.00 10.07 10.04 10.11 10.02 10.01 10.07 10.09 10.00 10.03 10.09 10.03 10.11 10.02 10.03 10.08 10.03 10.15 10.07 9.54 9.50 9.51 9.62 9.53 9.56 9.78 9.51 9.52 9.52 9.51 9.58 9.56 9.53 9.56 9.55 9.54 9.58 9.55 9.51 9.62 9.62 9.53 9.73 9.61 9.55 9.54 9.58 9.56 9.57 9.59 9.56 9.52 9.54 9.51 9.52 9.54 9.53

Les opérations binaires entre MDD

```
select tiff((char)((((unsigned short)a+
     (unsigned short)b)/2d)+0.5d))
from stopcol as a, spotcol as b
where oid(a)=15361 and oid(b)=15872
```

0.45 0.44 0.45 0.51 0.49 0.45 0.54 0.54 0.45 0.48 0.47 0.52 0.46 0.47 0.45 0.58 0.44 0.53 0.45 0.48 0.45 0.54 0.46 0.49 0.45 0.47 0.45 0.44 0.44 0.48 0.44 0.53 0.45 0.47 0.45 0.59 0.44 0.51 0.44 0.47 0.46 0.54 0.44 0.51 0.53 0.51 0.44 0.51 0.44 0.57 0.44 0.54 0.44 0.48 0.45 0.51 0.45 0.51 0.45 0.49 0.45 0.5 0.45 0.53 0.45 0.47 0.44 0.5 0.54 0.61 0.44 0.5 0.44 0.51 0.44 0.51 0.45 0.47 0.45 0.53 0.45 0.5 0.44 0.57 0.44 0.54 0.45 0.47 0.44 0.51 0.45 0.5 0.45 0.52 0.44 0.54 0.45 0.48 0.44 0.49

Masquage binaire et reclassification

```
select tiff(mdd > 100)
from mns as mdd
```

0.27 0.27 0.26 0.32 0.31 0.26 0.26 0.29 0.26 0.26 0.3 0.27 0.27 0.29 0.27 0.26 0.3 0.27 0.27 0.28 0.26 0.26 0.3 0.27 0.32 0.28 0.26 0.26 0.31 0.26 0.27 0.29 0.27 0.26 0.3 0.27 0.26 0.29 0.27 0.26 0.29 0.27 0.26 0.29 0.26 0.26 0.3 0.26 0.27 0.29 0.26 0.26 0.29 0.27 0.27 0.3 0.27 0.27 0.28 0.3 0.26 0.3 0.26 0.27 0.29 0.27 0.26 0.3 0.26 0.27 0.3 0.27 0.26 0.33 0.27 0.27 0.29 0.26 0.27 0.3 0.26 0.27 0.28 0.26 0.26 0.3 0.27 0.27 0.29 0.26 0.27 0.29 0.27 0.26 0.3 0.27 0.27 0.29 0.28 0.27

```
select tiff(mdd * (mdd > 100))
from mns as mdd
```

0.51 0.56 0.5 0.55 0.63 0.53 0.53 0.54 0.51 0.54 0.51 0.53 0.51 0.53 0.51 0.54 0.51 0.52 0.51 0.54 0.51 0.52 0.51 0.58 0.51 0.55 0.51 0.53 0.51 0.53 0.51 0.55 0.51 0.53 0.51 0.59 0.51 0.53 0.51 0.53 0.51 0.54 0.51 0.53 0.51 0.53 0.51 0.56 0.51 0.53 0.51 0.55 0.5 0.53 0.51 0.52 0.54 0.55 0.5 0.54 0.51 0.52 0.5 0.54 0.51 0.57 0.51 0.59 0.51 0.54 0.51 0.54 0.51 0.54 0.51 0.53 0.51 0.53 0.51 0.53 0.5 0.53 0.51 0.53 0.5 0.53 0.55 0.53 0.51 0.55 0.51 0.54 0.55 0.53 0.51 0.53 0.51 0.54 0.51 0.57

```
select tiff(
     (mdd =>100 and mdd < 200)
     + (mdd => 200 and < 300) * 2c
from mns as mdd
```

0.42 1.31 0.37 0.37 0.38 0.48 0.39 0.37 0.38 0.37 0.37 0.4 0.37 0.38 0.42 0.38 0.4
0.37 0.46 0.37 0.37 0.38 0.39 0.4 0.37 0.37 0.48 0.38 0.38 0.38 0.4 0.37 0.38 0.38
0.39 0.4 0.37 0.4 0.38 0.39 0.38 0.37 0.4 0.42 0.37 0.38 0.39 0.37 0.38 0.39 0.38
0.37 0.37 0.38 0.38 0.37 0.39 0.37 0.38 0.43 0.38 0.38 0.37 0.39 0.37 0.38 0.38
0.38 0.39 0.38 0.38 0.39 0.37 0.39 0.37 0.39 0.38 0.38 0.39 0.37 0.38 0.4 0.38 0.44
0.42 0.38 0.38 0.38 0.38 0.38 0.4 0.38 0.37 0.39 0.38 0.4 0.41 0.39 0.37 0.38

Mosaïquage

```
select tiff(
    extend(a, [0:1023, 0:1099]) overlay
        extend(shift(b,[0,500]),[0:1023, 0:1099]))
from landsatcol as a, landsatcol as b
where oid(a)=13313 and oid(b)=13825
```

0.49 0.46 0.45 0.52 0.48 0.59 0.57 0.48 0.75 0.62 0.48 0.47 0.48 0.47 0.49 0.46
0.48 0.45 0.47 0.46 0.49 0.45 0.48 0.46 0.47 0.73 0.54 0.46 0.57 0.45 0.48 0.48
0.47 0.45 0.49 0.45 0.49 0.46 0.5 0.45 0.48 0.46 0.49 0.45 0.48 0.46 0.49 0.46 0.48
0.44 0.48 0.45 0.48 0.45 0.49 0.45 0.49 0.45 0.68 0.46 0.48 0.45 0.48 0.46 0.5 0.45
0.47 0.46 0.48 0.46 0.48 0.46 0.48 0.46 0.48 0.45 0.49 0.45 0.46 0.44 0.5 0.46 0.46
0.44 0.5 0.45 0.47 0.46 0.49 0.45 0.48 0.65 0.46 0.48 0.52 0.48 0.45 0.47 0.46 0.48

Changement d'échelle

```
select tiff(
    scale(mdd, 2))
from landsatcol as mdd
```

1.84 2.98 1.9 1.85 1.86 2.43 1.96 1.87 1.86 1.85 1.86 1.86 1.86 1.86 2.6 1.86 1.86
1.88 1.87 1.86 1.87 1.86 1.85 1.86 2.15 1.97 1.86 1.87 1.86 1.86 1.87 1.91 1.85
2.59 1.86 1.86 1.88 1.86 1.86 1.86 1.87 1.87 1.86 2.34 1.95 1.86 1.87 1.85 1.87
1.85 1.85 1.88 2.72 1.86 1.87 2.07 1.87 1.85 1.86 1.91 1.85 2.76 1.98 1.87 1.86
1.86 1.87 1.86 1.86 1.86 2.78 1.88 1.85 1.86 1.85 1.86 1.87 1.85 1.87 1.85 1.86
1.98 1.87 1.86 1.86 1.88 1.87 1.9 1.86 2.5 1.86 1.87 1.87 1.85 1.85 1.86 1.85 1.87
1.86 2.56

Amélioration de contraste

```
select tiff(
    marray x in sdom(mdd)
        values (char)(((mdd[x[0], x[1]]) - 14)/180d)*255))
from spotcol as mdd
where oid(mdd)=15361
```

4.96 4.81 4.75 5.25 4.65 4.58 4.75 5.12 4.75 4.59 4.75 4.71 4.94 4.59 4.76 4.6 5.5
4.58 5.06 4.72 5.13 4.59 4.76 4.59 5.56 4.59 4.74 4.59 5.52 4.59 4.75 4.61 5.61
4.61 4.74 4.6 5.64 4.61 5.23 5 5.12 4.6 4.78 5.31 4.77 4.6 4.77 5.24 4.86 4.59 4.75
5.22 4.76 4.61 4.75 5.27 4.84 4.63 4.77 5.46 4.69 4.61 4.75 5.73 4.82 4.64 4.76
5.49 4.69 4.62 4.75 5.44 4.7 4.54 4.69 5.38 4.61 4.54 4.69 5.42 4.84 4.57 4.69 5.4
4.63 4.55 4.7 5.42 4.67 4.64 4.71 5.4 4.75 4.66 4.76 5.42 4.67 4.7 5 5.43

Histogrammes

Univariés

```
select
    marray x in [0:255]
        values count_cells(mdd=x)
from landsatcol as mdd
```

19.51 19.32 18.97 19.13 19.01 19.04 19.1 19.12 19.02 19.07 19.05 19.3 19.17 19.28 19.22 19.06 18.8 19.28 19.06 19.11 19.19 19.14 19.19 19.08 19.19 19.06 19.11 19.09 19.05 19.04 19.11 18.97 19.34 19.25 19.12 18.96 19.11 18.99 19.18 18.75 19.03 19.04 19 19.26 18.9 19.74 19.07 19.14 19.11 19.1 19 19.35 19.07 19.09 19.2 19.21 19.04 19.29 21.01 19.39 19.13 19.23 19.11 19.25 19.32 19.11 19.24 19.18 19.16 19.15 19.39 19.13 19.13 19.16 19.27 19.2 19.53 19.42 31.82 19.3 18.38 18.5 18.34 19.21 19.35 19.43 18.7 18.73 18.79 18.93 18.63 18.68 18.69 18.62 18.76 19.06 18.74 18.36 18.82 18.35

```
select
    marray x in [ min_cells(mdd):max_cells(mdd) ]
         values count_cells(mdd=x)
from landsatcol as mdd
```

18.78 19.17 18.93 19.16 18.57 18.82 18.74 18.67 19.08 18.96 19 18.8 19.43 19.02 18.73 18.82 18.91 18.9 18.93 19.11 20.75 19.22 18.87 18.8 18.82 18.8 18.81 18.57 18.89 19.04

Bivariés

Sur la collection `spotcol`

```
select tiff(
    marray x in [0:10, 0:10]
        values count_cells (a=x[0] and b=x[1]))
from spotcol as a, spotcol as b
where oid(a)=15361 and oid(b)=15873
```

9.77 9.79 9.76 9.8 9.82 10.42 10.54 9.79 9.86 10.61 9.79 9.77 9.78 10.58 9.75 9.76 9.75 9.88 9.76 12.02 9.96 10.85 9.97 10.8 10.07 10.82 10.79 9.86 11.04 9.74 11.21 9.75 11.55 9.78 11.41 10.02 11.2 9.92 11.16 9.98 11.36 11.49 9.76 11.86 9.81 11.63 10.02 11.57 9.93 11.57

```
select tiff(
    marray x in [0:25, 0:25]
        values count_cells (a=x[0] and b=x[1]))
from spotcol as a, spotcol as b
where oid(a)=15361 and oid(b)=15873
```

59.54 59.39 60.07 59.5 59.95 60.02 60.15 60.22 59.89 60.47 60.91 60.34 63.74 59.93 60.61 60.67 71.78 59.04 55.66 54.32 53.88 53.61 53.5 54.58 54.1 53.59 53.75 53.65 54.1 53.34 54.17 53.95 53.51 53.86 53.65 53.79 53.44 53.67 53.38 53.62 53.58 53.94 53.3 53.72 53.46 53.48 53.66 53.48 53.56 54.09

```
select tiff(
    marray x in [0:50, 0:50]
        values count_cells (a=x[0] and b=x[1]))
from spotcol as a, spotcol as b
where oid(a)=15361 and oid(b)=15873
```

196.19 194.74 194.45 194.54 194.32 194.22 194.11 194.55 194.19 194.35 194.85 194.6 193.96 193.97 193.77 193.83 194.01 193.58 193.68 193.55 193.28 193.32 193.46 193.64 193.15 193.33 193.45 193.23 193.11 193.51 193.08 193.25 193.44 192.99 193.25 193.43 193.09 192.99 193.45 192.95 192.88 193.35 193.05 193.07 193.42 193.08 193.04 193.47 193.03 193.06

Sur la collection `spot3`

```
select tiff(
    marray x in [0:10, 0:10]
        values (char)count_cells (a.0=x[0] and a.1=x[1]))
from spot3 as a
```

12.71 12.81 12.77 13 12.91 12.76 12.99 12.68 13.32 13.13 12.71 12.86 13 12.72 12.91 12.99 12.97 12.88 13.59 12.75 12.76 12.89 13 13.08 12.89 12.84 12.79 12.87 12.84 13.18 12.78 13.12 12.87 12.78 12.83 12.86 12.79 13 12.83 13.09 12.84 13.09 12.77 12.85 12.84 12.82 12.81 12.95 12.79 13.02

```
select tiff(
    marray x in [0:25, 0:25]
        values (char)count_cells (a.0=x[0] and a.1=x[1]))
from spot3 as a
```

71.38 71.34 71.06 71.45 71.45 71.24 71.3 71.35 71.22 71.27 71.21 71.25 71.32 71.17 73.31 72.05 72.21 71.94 72.27 72.37 72.35 72.05 72.51 72.55 72.12 72.45 72.14 72.4 72.18 72.24 72.32 72.23 72.39 72.47 71.79 72.37 72.18 72.33 74.15 72.39 72.88 72.78 71.95 72.71 72.47 71.87 72.69 72.03 72.61 72.32

```
select tiff(
    marray x in [0:50, 0:50]
        values (char)count_cells (a.0=x[0] and a.1=x[1]))
from spot3 as a
```

277.96 280.53 280.17 279.48 279.35 280.02 279.28 279.33 280.25 276.95 259.23 259.35 259.84 259.31 259.72 258.68 258.25 257.85 258.24 297.88 278.23 258.87 258.65 258.36 258.05 258.42 258.01 258.12 258.25 257.52 256.95 256.9 257.21 256.94 256.98 257.05 256.86 257.15 257.94 258.84 258.18 258.21 282.35 273.04 261.84 262.76 263.61 263.41 263.14 261.96

Les fenêtres de convolution

Le filtre moyen

```
select tiff(
    marray x in [sdom(mdd)[0].lo + 1: sdom(mdd)[0].hi - 1,
        sdom(mdd)[1].lo + 1: sdom(mdd)[1].hi - 1]
    values condense +
```

```
        over y in sdom(filtre)
        using (mdd[ x[0] + y[0], x[1] + y[1] * filtre[y]))
    from mns as mdd, filtre_moy as filtre
```

10.67 10.65 10.6 10.52 11.05 10.59 10.62 10.48 10.59 10.62 10.56 10.62 10.63 10.57 10.62 10.59 10.55 10.55 10.59 10.55 10.59 10.55 10.6 10.53 10.52 10.59 10.45 10.41 10.48 10.44 10.46 10.37 10.37 10.35 10.43 10.35 10.41 10.34 10.32 10.3 10.33 10.3 10.32 10.33 10.36 10.29 10.96 10.41 10.47 10.48

```
    select tiff(
        marray x in [sdom(mdd)[0].lo + 1: sdom(mdd)[0].hi - 1,
            sdom(mdd)[1].lo + 1: sdom(mdd)[1].hi - 1]
        values avg_cells(mdd[ x[0] - 1 + x[0] +1,
            x[1] - 1 + y[1] +1]))
    from mns as mdd
```

22.27 22.85 21.81 21.68 21.76 21.86 21.92 21.93 21.93 22.01 22.19 22.04 22.09 22.22 22.19 22.14 22.3 22.18 22.12 22.28 22.52 22.32 22.21 22.42 22.35 22.27 22.39 22.39 22.32 22.51 22.6 22.15 22.45 22.04 22.04 22.48 21.98 22.13 22.13 22.07 22.09 22.12 22.07 22.06 22.01 22 22.11 22.38 22.17 22.46

Le calcul de pentes

```
    select tiff((180/arccos(-1))*arctan(sqrt(
        marray x in [sdom(mdd)[0].lo + 1: sdom(mdd)[0].hi - 1,
             sdom(mdd)[1].lo + 1: sdom(mdd)[1].hi - 1]
        values condense +
        over y in sdom(gx)
        using (mdd[ x[0] + y[0], x[1] + y[1] * gx[y])
            *(mdd[ x[0] + y[0], x[1] + y[1] * gx[y])
            +(mdd[ x[0] + y[0], x[1] + y[1] * gy[y])
            *(mdd[ x[0] + y[0], x[1] + y[1] * gy[y]))))
    from mns as mdd, gradx as gx, grady as gy
```

197.4 196.8 196.56 200.17 197.42 198.28 197.4 198.05 197.97 197.32 199 197.38 196.86 197.14 198.06 196.58 197.63 196.82 196.78 197.88 196.53 196.66 198.17 198.09 198.49 196.6 196.76 196.17 195.73 197.2 196.5 197.42 200.89 197.74 197.08 196.81 196.65 197.37 196.85 198.16 197.13 197.29 197.69 196.95 198.17 197.56 197.8 197.41 199.36 198.37

Les requêtes

```
    select count_cells( mdd > 100 )from mns as mdd
```

0.22 0.14 0.14 0.14 0.15 0.14 0.15 0.14 0.15 0.14 0.15 0.14 0.15 0.15 0.14 0.14 0.14 0.14 0.14 0.15 0.14 0.14 0.15 0.14 0.15 0.14 0.14 0.14 0.15 0.15 0.14 0.15 0.24 0.17 0.23 0.23 0.27 0.24 0.23 0.22 0.24 0.15 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.16 0.18 0.24 0.24 0.23 0.22 0.24 0.15 0.14 0.15 0.2 0.22 0.14 0.14 0.14 0.15 0.14 0.15 0.14 0.14 0.14 0.16 0.17 0.16 0.14 0.14 0.14 0.15 0.19 0.15 0.14 0.15 0.14 0.15 0.14 0.15 0.14 0.14 0.14 0.14 0.15 0.15 0.16 0.14 0.14 0.14 0.15 0.14

```
select oid(mdd)
from spotcol as mdd
where some_cells(mdd < 20 )
```

0.24 0.25 0.24 0.24 0.25 0.25 0.25 0.25 0.25 0.24 0.24 0.26 0.25 0.24 0.25 0.24
0.25 0.25 0.24 0.24 0.25 0.25 0.25 0.25 0.26 0.25 0.25 0.24 0.24 0.24 0.25 0.25 0.5
0.42 0.25 0.38 0.43 0.33 0.25 0.25 0.24 0.25 0.24 0.24 0.24 0.44 0.4 0.35 0.25 0.25
0.25 0.24 0.24 0.24 0.24 0.28 0.24 0.24 0.25 0.24 0.24 0.24 0.25 0.24 0.24 0.24
0.25 0.24 0.24 0.24 0.24 0.25 0.25 0.25 0.24 0.25 0.24 0.25 0.24 0.25 0.24 0.23
0.24 0.25 0.25 0.24 0.26 0.24 0.24 0.25 0.25 0.24 0.25 0.24 0.25 0.25 0.27 0.25
0.25 0.25