Département de formation doctorale en informatique **École doctorale IAE + M**
UFR STMIA

Un environnement pour la reconstruction 3D d'édifices à partir de plans d'architecte

THÈSE

présentée et soutenue publiquement le 30 juin 2000

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

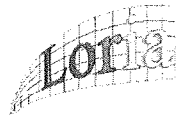
par

Philippe DOSCH

Composition du jury

Présidente et rapporteur :	Noëlle CARBONELL
Rapporteurs :	Rolf INGOLD
	Yves LECOURTIER
Examinateurs :	Josep LLADÓS
	Karl TOMBRE
Invité :	Michel MARS

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503



Résumé

Cette thèse s'inscrit dans le domaine de l'analyse de documents, et porte plus précisément sur la reconnaissance de graphiques. Notre objectif est d'obtenir un modèle 3D d'un édifice à partir de ses plans d'architecte. Pour cela, nous avons intégré des modules d'analyse et une interface homme-machine (IHM) permettant à l'opérateur de contrôler les traitements à effectuer de manière optimale. La majeure partie de la thèse détaille les différents traitements mis en œuvre, du bas-niveau (sur des images de plans numérisés) jusqu'au haut-niveau (sur des données vectorisées). Nous décrivons les choix d'architecture logicielle qui nous ont menés à définir un système composé de trois couches hiérarchiques : une bibliothèque de composants logiciels, une couche applicative regroupant les différents outils de traitement d'images et de graphiques et l'IHM. Celle-ci permet d'interagir directement sur les données, de contrôler le déroulement de l'analyse et gère le dialogue homme-machine. Dans ce travail d'équipe, nos contributions principales portent sur l'organisation spatiale des traitements (tuilage), l'extraction d'indices de niveau intermédiaire (lignes tiretées, symboles tels que les cages d'escalier...), la mise en correspondance des étages pour la reconstruction 3D du bâtiment correspondant, l'intégration logicielle et la mise au point de tout l'aspect IHM.

Mots-clés: reconnaissance de graphiques, plans architecturaux, analyse de documents techniques, architecture logicielle, interface interactive, reconstruction 3D

Abstract

This thesis is in line with the field of document analysis, and more precisely deals with graphics recognition. Our purpose is the construction of a 3D model of a building from the architectural drawings of its floors. For that, we have a set of analysis modules and a graphical user interface (GUI) allowing a human operator to control the processings to be performed in an optimal way. The major part of this thesis describes the various processings implemented, from the low-level (bitmap images processings) to the high-level (vectorized data processings). We describe the choices which have led us to define a three-layered software architecture, hierarchally organized: A library of software components, an applicative layer grouping the various processings together and the GUI. The latter allows to directly interact on data to control the the analysis, and manages the man-machine cooperation. All the members of our research teams have been involved in this work, but our main contributions concern the design of the GUI, the spatial organization of processings (tiling), the extraction of middle-level features (dashed and dotted lines, symbols such as stairwell, etc.) and matching algorithms to construct the 3D structure of a building, as well as the software integration and the design of the GUI.

Keywords: graphics recognition, architectural drawings, technical documents analysis, software architecture, graphical user interface, 3D reconstruction

Remerciements

Je souhaite tout d'abord remercier le CNET / France Telecom pour la confiance qu'il a placé dans le contrat qui a permis de financer cette thèse et, à travers cet organisme, Michel Mars qui a été notre interlocuteur dans le cadre de ce contrat. Je tiens à remercier tout particulièrement les personnes qui m'ont fait l'honneur de participer à mon jury :

- Noëlle Carbonell, professeur à l'université Henri Poincaré, rapporteur de ma thèse, qui a accepté très tôt de me relire et de me donner des recommandations,
- Rolf Ingold, professeur à l'université de Fribourg, qui a accepté de rapporter cette thèse,
- Yves Lecourtier, professeur à l'université de Rouen, rapporteur de mon travail,
- Josep Lladós, *Associate Professor* à l'université de Barcelone, d'avoir accepté d'être examinateur de ma thèse sur un sujet très proche de ses travaux,
- Karl Tombre, professeur à l'École des Mines de Nancy, qui a été mon directeur de thèse pendant les quatre années passées. Je lui suis tout particulièrement reconnaissant d'avoir accepté cette responsabilité et d'avoir orienté mon travail avec dynamisme et discernement. Qu'il trouve ici la marque de ma profonde reconnaissance, aussi bien au niveau professionnel que personnel.

Mes remerciements s'adressent également aux personnes avec qui j'ai travaillé, et en premier lieu à Christian Ah-Soon (alias Cracotte) avec qui j'ai travaillé très étroitement. Je le remercie tout particulièrement d'avoir accepté de relire une version préliminaire de ma thèse en y apportant des remarques pertinentes. Je remercie également Gérald Masini, dont les qualités de rigueur et de concision m'ont beaucoup apporté. Enfin, je remercie également tous les membres de l'équipe ISA avec qui j'ai passé quatre années très agréables et instructives. Je remercie particulièrement Sylvain Petitjean, pour sa disponibilité sur les divers problèmes techniques auxquels il a toujours une réponse à apporter, Marie-Odile Berger pour ses éclaircissements en mathématiques et Brigitte Wrobel-Dautcourt pour ses coups de pouces en script (à force de lire les remerciements de thèses, on fini par savoir à qui s'adresser ☺).

Je remercie les nombreux soutiens que j'ai trouvés en dehors de l'équipe. Je tiens ainsi à remercier particulièrement Alain Filbois, pour ses dépannages express et pour les causettes le soir pendant ma rédaction, ainsi que Denis Roegel pour sa connaissance et sa disponibilité sur LaTeX, et pour la classe `thloria` qu'il a conçu. Enfin, je tiens à remercier l'ensemble des Moyens Informatiques pour leur compétence et la qualité du système informatique qu'ils entretiennent au LORIA.

Enfin, si de bonnes conditions de travail sont conditionnées par un bon environnement professionnel, elles le sont aussi par un bon environnement personnel. À ce titre, je souhaite tout d'abord remercier ma famille et particulièrement mes grands-parents, mes parents et beaux-parents, ainsi que mes sœurs. Je souhaite également saluer mes amis, les gens de Phi-Sciences, les Gunchos, les exilés de Nancy, qu'ils se trouvent en France, outre-Manche ou outre-mer, et tout ceux que je connais de près ou de loin qui se reconnaîtront.

Table des matières

Table des figures

Introduction

Après avoir exploré l'espace en deux dimensions, l'informatique graphique s'est résolument portée vers la troisième dimension. Plusieurs facteurs ont contribué à cette nouvelle quête : ceci n'aurait en particulier pas été possible sans les performances toujours plus accrues des ordinateurs, qui ont longtemps constitué le goulot d'étranglement de ce type de recherche. Mais cela répond surtout au besoin compréhensible de se doter d'outils capables d'appréhender le monde réel dans lequel nous vivons. Il n'est ainsi plus rare de nos jours d'être en contact avec la 3D, que ce soit par l'intermédiaire de jeux, de navigations virtuelles, jusqu'aux systèmes de fenêtrage qui commencent maintenant à intégrer cette dimension supplémentaire. Des applications plus professionnelles exploitent également la 3D, à des fins de simulation, de visualisation...

À ce stade, on peut différencier deux types d'applications 3D : celles basées sur un modèle et celles qui ne le sont pas. La création du modèle, lorsqu'il est nécessaire, peut s'effectuer par l'utilisation de techniques telles que la CAO[1]. Si ce procédé est efficace pour la définition complète d'un modèle — la conception d'un nouveau bâtiment par exemple —, il est cependant peut adapté à l'exploitation de données existantes — les plans d'architecte d'un bâtiment par exemple. En fait, dans ce dernier cas, il est nécessaire de mettre en œuvre des techniques adaptées, permettant de « récupérer » sous une forme informatique (ou électronique) les documents existants, ou ceux qui continuent d'être produits sous une forme manuelle.

Des exemples concrets de ce type de techniques sont d'ailleurs en application depuis plusieurs années. C'est grâce à ces procédés que la poste trie le courrier, que les entreprises de vente par correspondance analysent aujourd'hui les bons de commande de leurs clients et que les banques traitent leurs chèques. De même, tous les scanners disponibles sur le marché aujourd'hui incluent un logiciel de reconnaissance de caractères (OCR[2]), ce qui indique d'une part que ce type de logiciel est arrivé à maturité, et d'autre part qu'il correspond à une attente de la part des acquéreurs de scanner.

D'autres applications existent, même si elles sont moins connues du grand public. On trouve parmi elles des applications qui se sont intéressées à l'analyse de documents techniques, au sens large du terme. Les besoins de ce domaine sont en effet importants et regroupent divers secteurs d'activités. Il y a ainsi des besoins en cartographie, en gestion de plans cadastraux, en électronique ou mécanique, en édition de partition musicale... Parmi ces nombreux domaines, notre équipe s'est positionnée il y a quelques années sur l'analyse de documents architecturaux, après avoir travaillé sur d'autres types de documents techniques.

C'est dans ce contexte qu'un partenariat a été établit entre notre équipe et le CNET[3]. En effet, le CNET s'est intéressé à la modélisation de la propagation d'ondes hertziennes pour répondre à des problèmes

1. Conception Assistée par Ordinateur.
2. *Optical Character Recognition.*
3. Centre National d'Étude des Télécommunications.

de transmission téléphonique, notamment avec les téléphones cellulaires dans de grands édifices publics. Après avoir expérimenté la modélisation des bâtiments à étudier par saisie informatique manuelle à l'aide de tables à numériser, il a souhaité se tourner vers un système :

- permettant d'appliquer un enchaînement d'algorithmes afin d'obtenir une reconstruction 3D d'un bâtiment à partir de ses plans d'architecture, généralement disponibles,
- incluant des outils ergonomiques pour permettre à un utilisateur de disposer de certaines interactions, notamment des possibilités de reprise et d'affinage du modèle 3D généré.

L'objectif de ce contrat, formalisé par un cahier des charges du système à réaliser, était multiple :

- adapter les traitements développés par l'équipe en analyse de plans architecturaux aux besoins spécifiques du CNET,
- développer des traitements supplémentaires répondant aux exigences du cahier des charges établi,
- encapsuler l'intégralité des traitements développés au sein d'une IHM[4] permettant à l'utilisateur de suivre le déroulement de l'analyse d'un plan et d'interagir avec les résultats produits,
- assurer l'intégration du système produit au sein de l'architecture logicielle du CNET, notamment par le formatage des entrées-sorties et le portage sur une architecture donnée.

C'est dans ce contexte que la présente thèse a été définie. Les différentes étapes nécessaires à la réalisation du contrat passé avec le CNET y sont abordées, resituées parmi les problèmes plus généraux relatifs à l'analyse de documents techniques. Ceci inclut les aspects scientifiques liés au système d'analyse de plans et les aspects plus pratiques liés à la nécessité de produire un environnement permettant de suivre toute la progression de l'analyse. Ce document s'articule autour de cinq grandes parties thématiques, décomposées en chapitres, permettant de distinguer les différents aspects abordés, comme le passage de la théorie à la pratique, ou encore la part des travaux réalisés de façon communautaire des travaux plus personnels.

La première partie est une partie introductive, préambule à la chaîne des traitements appliqués lors de l'analyse. Le chapitre 1 introduit la notion même d'analyse de documents techniques, en se basant sur quelques exemples et sur les plans architecturaux en particulier. Le chapitre 2 présente les principes retenus et la terminologie utilisée dans ce document ; il fait ressortir en particulier une grille de lecture quant à la méthodologie suivie pour l'élaborer.

La deuxième partie débute la description des traitements appliqués pour analyser les plans architecturaux, en se focalisant sur les traitements dits de « bas-niveau », concernant le plan lorsqu'il est manipulé sous forme d'image *bitmap*[5]. Le chapitre 3 décrit les techniques retenues par notre équipe pour effectuer ces traitements bas-niveau. Le chapitre 4 introduit la notion de *tuilage*, technique qui permet l'analyse de grands documents, qui seraient sinon difficilement manipulables sous cette forme sur des stations de travail classiques.

La troisième partie se focalise sur le problème particulier de la *vectorisation*, c'est-à-dire sur la technique permettant de convertir une image *bitmap* en un ensemble de *vecteurs*, primitives graphiques de base entrant dans la composition des symboles architecturaux recherchés. Le chapitre 5 présente les techniques retenues par notre équipe pour effectuer la vectorisation d'une image numérisée. Le chapitre 6 s'intéresse à un type de primitives fréquemment rencontrées sur les documents techniques et sur les plans architecturaux en particulier, les lignes tiretées. Le chapitre 7 décrit la technique utilisée pour extraire un autre type de primitives graphiques, les arcs de cercles, à partir des segments extraits et des données temporaires générées par l'étape de vectorisation.

4. Interface Homme-Machine.

5. C'est-à-dire sous forme d'image de points. Nous utilisons également le terme *image numérisée* pour désigner ce type d'image.

La quatrième partie de ce document introduit les différentes techniques utilisées pour générer un modèle 3D d'un bâtiment à partir de sa description en termes de primitives graphiques. Le chapitre 8 dresse un état de l'art en reconnaissance de symboles. Nous présentons ensuite chapitre 9 un résumé des travaux réalisés par Christian Ah-Soon sur la détection de symboles architecturaux par utilisation d'un réseau de contraintes. Le chapitre 10 est dédié à la recherche des cages d'escaliers, symbole architectural non considéré dans le chapitre précédent. À l'issue de ce chapitre, un modèle 2D de chacun des niveaux du bâtiment étudié est calculable, ainsi que le modèle 3D correspondant; ces étapes sont décrites chapitre 11. Les techniques utilisées pour recaler les modèles 3D des différents étages sont détaillées chapitre 12.

Enfin, la cinquième partie présente les choix retenus pour la mise en œuvre de l'environnement d'analyse créé et notamment sur l'intégration de tous les traitements décrits au sein d'un système interactif. Le chapitre 13 décrit les contraintes imposées par une telle interface, étudie les différentes approches réalisables et présente finalement l'architecture logicielle retenue pour intégrer cette interface. Le chapitre 14 présente la mise en œuvre des deux premières composantes de notre système, qui correspondent aux traitements liés à l'analyse de documents. Le chapitre 15 décrit l'interface interactive réalisée dans ce contexte, en présentant ses fonctionnalités et la démarche générale adoptée lors de sa mise en œuvre.

Le document se conclut par une réflexion sur les travaux réalisés et sur les perspectives envisageables, ainsi que par des annexes techniques relatives notamment aux bibliothèques utilisées pour développer l'interface homme-machine. Dans ce travail d'équipe, nos contributions principales portent sur des aspects d'organisation spatiale des traitements (tuilage), d'extraction d'indices de niveau intermédiaire (lignes tiretées, arcs, symboles tels que les cages d'escalier...), de mise en correspondance (reconstruction 3D par appariement d'étages), d'intégration logicielle et de mise au point de tout ce qui touche à l'interface homme-machine.

Première partie

Contexte

Chapitre 1

Analyse de documents techniques

1.1 Une large variété de documents techniques

Les documents techniques, employés dans la majeure partie des domaines de l'ingénierie, permettent de communiquer un savoir. Ce sont des documents essentiellement graphiques pouvant provenir de différentes sources : dessinateur, système de CAO. Adaptés à leur domaine d'utilisation, ils respectent généralement des normes de représentation assez strictes qui permettent de représenter de façon compacte une information détaillée d'un produit, d'un ouvrage, tout en évitant une quelconque ambiguïté. La tendance actuelle pour la conception de tels documents est d'utiliser des solutions informatiques. En effet, de nombreux logiciels de CAO existent, proposant des outils de conception adaptés à chaque domaine. Cette approche facilite le stockage, la manipulation, la modification ou la recherche des documents. C'est pour cette raison que des efforts ont été entrepris pour convertir les documents techniques disponibles sur un support papier en une représentation informatique. Cette opération est dénommée *analyse de documents techniques*.

Au vu de la variété de domaines utilisant les documents techniques, et du nombre très important de documents à convertir, de multiples solutions ont été proposées pour traiter ce problème, y compris des solutions commerciales. La majorité des systèmes commerciaux procèdent selon le même schéma :

- la numérisation du document initial à l'aide d'un scanner,
- l'application de certains filtres destinés à éliminer le bruit du document original,
- la segmentation de la partie texte et de la partie graphique,
- la reconnaissance des parties textuelles au moyen d'un OCR,
- la vectorisation de la partie graphique (c'est-à-dire la conversion de l'image en un ensemble de vecteurs),
- la reconnaissance de symboles simples.

On peut cependant noter qu'aucun de ces systèmes ne permet d'apporter une solution complètement fiable, et qu'il est nécessaire de reprendre manuellement tout ou partie des résultats issus des différentes

phases énumérées. Ils présentent notamment le défaut d'être trop généraux ou de nécessiter une phase de réglage trop importante et trop délicate de leurs différents paramètres.

Au niveau de la recherche scientifique, il existe également de nombreux travaux, qui ont conduit à l'élaboration de systèmes adaptés au type de documents à traiter. Nous énumérons ci-dessous certains des domaines explorés, en les accompagnant de quelques références récentes ou significatives, et en présentant certaines de leurs caractéristiques.

- Les *plans électroniques*, qui ont engendré de nombreuses publications [Fah 88, Lee 92, Zes 94, dJ 95]. Ceux-ci sont composés de symboles dont la représentation est normalisée, de lignes reliant ces symboles et d'annotations. Les recherches effectuées ont permis de mettre au point des systèmes utilisés dans les milieux industriels, en utilisant notamment des techniques à base d'isomorphisme de sous-graphes.
- Les *documents mécaniques* [Bot 95, Vax 95, Ram 96, Tom 96a]. Ici, les documents présentent des vues correspondant aux projections orthogonales réalisables selon chacun des axes d'un espace en trois dimensions. Les documents contiennent généralement des annotations, des axes de symétrie, des parties cachées représentées selon une convention particulière (souvent des lignes tiretées). La représentation de ce type de documents est fortement normalisée, ce qui permet d'injecter des connaissances *a priori* dans les systèmes de reconnaissance.
- La *cotation*, qui est un problème récurent en analyse de documents techniques [Dor 89, Col 92, Pri 96, Lin 97, Dor 98a]. En effet, des éléments de cotation peuvent se trouver sur des documents de domaines différents, mécaniques ou architecturaux par exemple. L'analyse de la cotation enrichit alors les informations extraites par l'analyse spécifique au document étudié. La cotation, généralement bien normalisée, fait ainsi l'objet d'un nombre important de publications.
- Les *cartes géographiques* [Sme 96, Sam 98, Rou 98], domaine dynamisé par la réalisation de SIG[6]. Une littérature très abondante est disponible sur ce sujet, les acteurs intéressés par ce domaine étant nombreux (instituts géographiques, militaires).
- Les *plans cadastraux* [dH 96, Jan 97a, Dos 97, Ogi 97, Lla 99]. Dans ce type de plans, l'analyse doit être particulièrement précise, le document ayant généralement valeur légale. De nombreux types de cadastres ont été étudiés (français, italien, hollandais, chinois...), la représentation variant d'un pays à l'autre.
- Les *partitions musicales* [Blo 92, Bai 97, Anq 99, Blo 99]. Même si ce type de documents n'est pas précisément un document technique au sens où nous l'avons introduit, ses caractéristiques de représentation le prédispose au même type d'analyse que les documents énumérés ci-dessus.
- Et bien d'autres, comme les *plans hydrauliques* [Fut 90], les *organigrammes* [Abe 86], les *plans de câblage téléphonique* [Ada 00], ...

Chaque type de documents évoqué est doté de particularités qui orientent l'analyse à mettre en œuvre pour en extraire les informations pertinentes. On peut cependant distinguer d'une manière générale trois étapes principales dans les processus d'analyse. Tout d'abord, une étape *lexicale* qui correspond aux étapes de bas-niveau (cf. deuxième partie) et de vectorisation (cf. troisième partie). À l'issue de cette étape, on dispose d'une description du document en terme de primitives vectorielles, qui sont utilisées dans une étape *syntaxique*, se basant sur des règles d'assemblage de ces primitives. Les méthodes mises en œuvre peuvent être structurelles et syntaxiques (cf. chapitre 8). Enfin, une étape *sémantique* peut être ajoutée pour contrôler la consistance des assemblages déterminés, selon des règles qui sont propres au domaine considéré.

6. Systèmes d'Information Géographique.

1.2 Les plans architecturaux

1.2.1 Caractéristiques

Les plans architecturaux constituent une catégorie de documents techniques et peuvent donc être traités au moyen des techniques utilisées pour les documents techniques énumérés au paragraphe précédent. Ils sont également dotés de caractéristiques qui les différencient des autres documents techniques. La plus importante d'entres elles est le fait qu'ils ne sont pas complètement normés. À ce titre, l'architecte dispose d'une certaine latitude de représentation des différentes composantes architecturales. Certaines conventions se dégagent toutefois, la représentation des symboles architecturaux étant souvent proche de leur apparence physique. On notera cependant qu'aucun standard ne s'est imposé, et que l'analyse de plans architecturaux nécessite de considérer des représentations hétérogènes de symboles, parfois sur un même plan.

La représentation d'un plan architectural peut s'apparenter à celle des documents techniques et, dans une certaine mesure, à celle d'un document artistique. L'élaboration d'un plan nécessite plusieurs phases, de la plus conceptuelle à la plus technique [Tom 95] :

- l'*esquisse*, qui correspond aux premières intentions de l'architecte. Dans cette phase, le trait est essentiellement artistique et ainsi la représentation assez libre ;
- l'*avant-projet*, qui comprend des plans, des vues en coupe, des élévations. C'est typiquement le type de plans présenté pour l'obtention des permis de construire. Au niveau représentation, les informations ne sont ni trop floues ni trop détaillées, et il y a beaucoup plus de rigueur qu'au niveau de l'esquisse ;
- les *plans d'exécution*, qui représentent les informations architecturales, mais aussi celles relatives aux autres corps de métier (électricité, plomberie, etc.) ainsi que des informations techniques : spécification de matériaux par exemple. Ce type de documents est le plus précis, mais aussi le plus dense au niveau des informations représentées.

Si les documents de la première phase sont trop conceptuels, ceux correspondant à la dernière phase sont souvent trop détaillés et chargés pour être exploitable par une analyse automatique. Les plans correspondant à l'avant-projet sont donc les plans privilégiés pour une analyse.

1.2.2 Étude des approches observées

Par rapport aux autres catégories de documents techniques, peu de travaux ont été publiés sur l'analyse de plans architecturaux. Nous présentons ci-dessous sommairement les références dont nous avons connaissance :

- Koutamanis [Kou 90, Kou 95] décrit un système permettant d'analyser un plan architectural, en y recherchant des entités architecturales telles que les murs, les colonnes, les fenêtres et les portes. À partir des entités reconnues, il détermine ensuite l'arrangement spatial du bâtiment. L'approche est cependant peu flexible, limitée par le choix des entités, ce qui suppose une standardisation de la représentation qui est rarement effective.
- Ryall et Shieber [Rya 95] proposent un système interactif semi-automatique pour déterminer les différentes pièces représentées sur un plan. Pour cela, ils utilisent une métrique qui permet de déterminer les régions plus ou moins closes du plan, ce qui revient intuitivement à remplir les espaces, c'est-à-dire les zones de points blancs. À l'issue de la détection automatique, l'utilisa-

teur peut interactivement corriger les résultats. Le système se limite à une détection spatiale, sans chercher à identifier les symboles architecturaux.

- Gross [Gro 95] présente un système d'indexation de plans architecturaux au moyen de critères géométriques simples. Les informations extraites des plans à référencer sont regroupées dans une base de données, qu'il est possible d'interroger en dessinant manuellement une esquisse du plan recherché. L'esquisse est analysée de la même manière que les plans de référence, puis confrontée aux plans de la base. Les critères d'indexation portent sur la forme extérieure du bâtiment, la disposition ou la forme des pièces. Là encore, l'objectif du système n'est pas d'extraire les composantes architecturales ; le système se concentre sur l'étude de caractéristiques discriminantes.
- Aoki *et al.* [Aok 96] décrivent un système d'interprétation de plans architecturaux manuels. Ils imposent une contrainte sur la conception de ces plans, qui doivent être dessinés en s'alignant sur le quadrillage d'une feuille, prévue à cet usage, en utilisant un stylo dont l'épaisseur de trait est fixée. De cette manière, ils limitent les problèmes de bruit sur le dessin original. Leur analyse débute par deux recherches : celle des lignes de l'image, qui s'appuie sur les contraintes énoncées, et celle des régions closes, déterminées par l'étude des zones de points blancs. Des modèles de région, correspondant à des formes géométriques simples, permettent de classifier les régions extraites. À partir de ces deux primitives, ils extraient les symboles architecturaux qui sont modélisés simplement. Leur taux de reconnaissance est très bon mais impose le respect des contraintes de conception adoptées.
- Lladós *et al.* [Lla 97b, Lla 98, Lla 00] s'intéressent également aux plans manuels. Ils y recherchent les symboles architecturaux « classiques » (portes, fenêtres, murs, etc.), ainsi que des symboles de mobilier tels que des lits, des tables... Deux types de symboles sont distingués : ceux dont la forme peut être décrite directement et ceux qui peuvent être reconnus par le type de texture qui les compose (principalement des hachures). Ils appliquent une vectorisation au dessin traité, qui génère une représentation de celui-ci sous la forme d'un graphe attribué, gérant des relations du type *à-côté-de*, *est-contenu-dans*, etc. Le premier type de symboles, modélisé également sous forme de graphe, est reconnu au moyen d'isomorphisme de sous-graphes. Le deuxième type est recherché au moyen d'une méthode basée sur la transformée de Hough, qui permet de caractériser le type de texture (directions dominantes, variations de direction). D'autres types de texture sont également étudiés, permettant de distinguer des tuiles, du carrelage [Sán 97] (cette méthode est détaillée plus loin [§ 10.3]).
- Sur le cas particulier de la reconnaissance de symboles architecturaux, Valveny et Martí [Val 99] présentent une approche à base de *template matching* pour contourner les problèmes inhérents au bruit. En effet, cette technique propose de manipuler directement l'image binaire, évitant l'étape de vectorisation et les artefacts qu'elle produit. Le *template matching* est en outre particulièrement bien adapté aux distorsions existant entre un modèle de symbole et les occurrences réelles de symboles qui lui correspondent. Un coût est associé à chaque type de variation (translation ou rotation d'une primitive par rapport à une autre par exemple). La reconnaissance de symboles est ensuite formulée selon une approche Bayesienne, consistant à minimiser une fonction de coût. Les résultat obtenus montrent la robustesse de l'approche, même si en raison de la nature des données (une image binaire), celle-ci s'applique essentiellement sur les symboles isolés, c'est-à-dire non connectés au reste du plan traité.

1.2.3 Synthèse

Il y a finalement peu de méthodes proposées pour l'analyse de plans architecturaux par rapport aux autres domaines du document technique. Les besoins de conversion de ce type de plans existent cepen-

dant, que ce soit de la part des architectes, ou plus majoritairement de la part des personnes qui ont besoin d'une modélisation 2D de bâtiment, à des fins de visualisation, de simulation... La difficulté principale à appréhender sur ce type de plans réside dans leur représentation. En l'absence de norme, il est en effet difficile de concevoir un système permettant de traiter n'importe quel type de plan architectural. Cette remarque suggère qu'un système d'analyse architectural doit pouvoir être flexible et adaptatif. Cette contrainte est parfois détournée sur les systèmes qui injectent des règles supplémentaires sur la représentation des plans, comme l'utilisation d'une grille par exemple. Ce type de contrainte ne peut cependant pas s'appliquer sur les plans déjà créés, ce qui ne résout pas tous les problèmes.

D'une manière générale, l'analyse de documents techniques ne permet pas d'obtenir des résultats pleinement satisfaisants et trouve ses limites sur des problèmes qui restent ouverts. De plus en plus fréquemment, les systèmes d'analyse sont assistés par l'utilisateur au moyen d'une interface graphique par laquelle il peut interactivement reprendre les données incorrectes. Dans notre contexte, ce problème est assez préoccupant, les défauts des traitements couramment utilisés en analyse de documents techniques se cumulant au manque de normalisation. Certains auteurs (Ryall et Shieber [Rya 95]) proposent d'ailleurs d'adjoindre une interface interactive à leur système d'analyse. Nous avons opté pour ce type d'approche.

En ce qui concerne la nature des plans architecturaux à considérer, nous avons souhaité nous placer dans un cadre le plus large possible, même si nous ne prétendons pas pouvoir analyser tous les types de plans disponibles. Cette orientation nous a amener à considérer des approches flexibles, qui puissent évoluer en fonction de la représentation utilisée pour un plan donné. Dans cette optique, nous nous sommes efforcés de nous doter de traitements robustes et réutilisables, pouvant s'accommoder des différences de représentation existantes. Ces principes ont servi de base à l'élaboration du système d'analyse de plans architecturaux présenté dans ce mémoire.

Chapitre 2

Principes de conception

2.1 Introduction

L'objectif de ce chapitre est de présenter la méthodologie de conception retenue pour l'élaboration de notre système. Cette méthodologie est guidée par les préoccupations liées à notre domaine de recherche et plus généralement à celles communes à tout chercheur en informatique. D'une part, la réalisation de systèmes informatiques, quelle qu'en soit la vocation, ne se conçoit plus sans l'intégration d'un certain nombre d'outils logiciels (environnements de programmation, bibliothèques logicielles, ...) Ces outils offrent beaucoup d'avantages, comme des possibilités de développement rapide, mais ne sont pas utilisés systématiquement ou sont parfois utilisés sans une bonne connaissance des concepts qu'ils véhiculent. D'autre part, l'analyse de documents a atteint une certaine maturité et les chercheurs de ce domaine souhaitent se doter d'un ensemble de traitements fiables et robustes, apportant des solutions génériques aux différents problèmes à considérer. La recherche de ce type de traitements est devenu une problématique à part entière, notamment dans des conférences telles que ICDAR[7] ou GREC[8] [Tom 98b]. Cela pose un problème plus vaste : celui de l'évaluation de performances, et ainsi de l'identification de critères qualitatifs permettant d'apprécier l'aptitude d'un traitement.

Il apparaît clairement que la conception et l'évaluation de performances deviennent des enjeux cruciaux pour la correction et l'évolutivité des systèmes créés. Ces deux aspects ont particulièrement retenu notre attention. Nous avons ainsi élargi notre problème d'analyse de documents techniques à celui de la mise en œuvre d'un système effectuant une telle tâche. Tout d'abord au niveau de la conception de nos traitements, pour lesquels nous nous efforçons de détailler les étapes allant de leur définition jusqu'à leur mise en œuvre. Ensuite, au niveau de la possibilité d'effectuer une évaluation de performances de ces traitements, selon des critères bien définis. Nous estimons qu'il est nécessaire d'intégrer ces caractéristiques au développement de nos travaux, afin de les resituer par rapport à notre domaine de recherche. Nous introduisons dans ce sens les principes de conception utilisés lors de l'élaboration de notre système.

7. *International Conference on Document Analysis and Recognition.*
8. *International Workshop on Graphics Recognition.*

2.2 Terminologie — Décomposition hiérarchique d'un traitement

La construction d'un système complet d'analyse de documents techniques nécessite la définition d'un certain nombre de traitements. La notion même de traitement peut être perçue à différents stades de conception. Afin de différencier chacun de ces stades, Smeulders et de Boer [Sme 98] ont introduit une terminologie permettant de décomposer hiérarchiquement la conception d'un traitement :

- La *méthode*, qui correspond au niveau mathématique et linguistique du traitement. On y trouve une définition des buts, des contraintes, des données et des résultats attendus, ainsi qu'une abstraction qui ne transparaît généralement pas au niveau de l'algorithme (niveau suivant). C'est le niveau généralement décrit dans la littérature scientifique.
- L'*algorithme*, c'est-à-dire la décomposition d'une méthode en étapes élémentaires dans un ordre logique. Cela nécessite le maintien d'une certaine abstraction pour conserver le caractère générique de la description, mais permet cependant de tenir compte d'autres facteurs, comme l'architecture logicielle par exemple. Les algorithmes sont plus rarement décrits dans la littérature, hormis lorsqu'ils présentent une solution originale, bien que le passage de la méthode à la mise en œuvre (niveau suivant) n'est pas toujours simple. Le coût ou la complexité des algorithmes sont parfois introduits, ce qui peut constituer un premier élément d'appréciation de la performance du traitement.
- L'*implémentation*[9], le *codage* ou la *mise en œuvre*[10], qui correspondent au niveau ultime de la conception. Ce niveau est dépendant du langage, de la machine, de l'environnement, etc., ce qui en fait la partie la moins persistante du traitement. Elle doit en effet suivre l'évolution des composantes dont elle dépend, ce qui oblige bien souvent à réécrire tout ou partie du code. Il est ainsi important à ce stade de produire du logiciel robuste et efficace, ce qui nécessite l'emploi de techniques d'optimisation de code, de génie logiciel et d'architecture logicielle.

Cette décomposition n'est pas complètement rigide : certaines applications peuvent entraîner l'utilisation de flux entre ces différents niveaux. Elle est cependant utile pour fixer un cadre de définition dans l'élaboration d'un traitement, en faisant ressortir les différents niveaux auxquels peuvent être comparés des traitements concurrents. Elle sert ainsi de base pour la définition de l'évaluation de performances.

2.3 Évaluation de performances

2.3.1 Analyse des approches réalisables

Nous détaillons dans ce paragraphe les principes inhérents à l'évaluation de performances, ainsi que quelques exemples concrets mis en œuvre. L'évaluation de performances est une problématique maintenant bien établie au sein de notre communauté, pour laquelle plusieurs voies ont été proposées, formalisant quelque peu cette technique. Tout d'abord, afin de pouvoir exploiter les résultats fournis par une évaluation de performances, il est nécessaire d'en définir un modèle. Quel que soit le modèle élaboré, l'objectif est d'estimer ou de prédire le comportement d'un traitement dans une situation concrète [Sme 98]. En fait, le type d'évaluation de performances à considérer dépend fortement du niveau de conception auquel on se situe.

9. Terme retenu par la commission ministérielle de terminologie informatique pour désigner la phase finale d'élaboration d'un système.
10. Nous utilisons indifféremment ces trois termes dans ce manuscrit.

Elle se conçoit à chacun des niveaux définis dans le paragraphe précédent selon des critères différents :

- pour les méthodes, l'évaluation peut reposer sur le nombre de paramètres et la dépendance liée à ces paramètres. En effet, plus une méthode comporte de paramètres, plus elle est délicate à adapter à un besoin précis. Si ces paramètres sont de plus interdépendants, cela complique d'avantage leur réglage. Pour les méthodes qui peuvent être modélisées — des données qu'elles acceptent jusqu'aux résultats qu'elles produisent —, l'évaluation peut consister à modéliser le bruit contenu dans des données et à mener une étude de la propagation de ce bruit. On mesure alors la robustesse de la méthode par l'influence de faibles bruits introduits sur les données par rapport aux variations perçues dans les résultats ;
- pour les algorithmes, l'évaluation peut être basée sur une estimation de leur complexité. Elle peut aussi être basée sur la vérification de la correction d'un algorithme, en déterminant son comportement sur un ensemble exhaustif des différentes données possibles ;
- pour le codage, il est possible d'effectuer l'évaluation à partir d'étude de la propagation d'erreur ou de l'utilisation de tests empiriques. Dans le premier cas, cela nécessite de trouver un ensemble d'erreurs représentatives pouvant intervenir à différents stades du traitement et une mesure statistique de la propagation de ces données. Le procédé est similaire sur le principe à ce qui est réalisable sur les méthodes, mais opère sur des données existantes et non plus théoriques. La mise en œuvre de cette technique est souvent facilitée par les nombreux outils disponibles (comme les *debuggers*), ce qui lui confère un avantage par rapport à l'étude analogue sur les méthodes. Dans le second cas, on considère les applications comme des boîtes noires testées sur des échantillons, qui peuvent être réels ou synthétiques. Les résultats obtenus sont alors comparés aux résultats idéaux à l'aide d'une mesure de distance. Cette approche est souvent la seule réalisable étant donnée la complexité des environnements et des algorithmes développés.

Toutes ces approches sont complémentaires : l'étude des méthodes opère à un niveau qui s'avère pertinent sur les méthodes qui sont bien modélisées. Il est ainsi rarement possible de le mettre en œuvre dans notre problématique. Au niveau des algorithmes, l'évaluation est plus facilement réalisable, mais nécessite une prise en compte exhaustive des données acceptées par l'algorithme. Ce n'est pas toujours possible par le biais de données réelles, mais les différents cas peuvent être approchés à l'aide de données synthétiques. En ce qui concerne le codage, dont la dépendance avec l'algorithme est très proche, le seul moyen raisonnable est l'utilisation de tests empiriques. Pour être valides, ces tests doivent être effectuées systématiquement sur des échantillons représentatifs, et peuvent ainsi dans une certaine mesure prouver la validité de la méthode.

Quelles que soient les approches observées, le soin apporté à chaque stade de la conception est essentiel pour la correction du traitement à réaliser. Cet investissement est généralement bénéfique, il permet de favoriser les performances des étapes suivantes de conception et ainsi des performances du traitement.

2.3.2 Quelques évaluation réalisées

Les évaluations liées à l'analyse de documents techniques ont essentiellement porté sur les traitements que nous qualifions de « bas-niveau » (cf. chapitre 3). C'est en effet sur ces traitements qu'un grand nombre de méthodes ont été proposées. Il y quelques années, Haralick [Har 92a] a ainsi proposé une évaluation de performances des méthodes de squelettisation par amincissements, les caractéristiques liées à ce type de méthodes étant relativement faciles à modéliser. D'autres travaux ont été réalisés sur

ce point, comme ceux de Jaisimha *et al.* [Jai 93]. Des évaluations ont également été conduites sur des méthodes de binarisation [Tri 95b] ou de segmentation [Ran 94].

Les évaluations de performances sont maintenant de plus en plus courantes, notamment dans les conférences auxquelles nous sommes affiliés, qui comportent régulièrement des tournois d'évaluation de logiciels. Ainsi, la conférence GREC'95 a été l'occasion de tester divers logiciels de détection de lignes pointillées à partir de données synthétiques. Une méthode d'évaluation de performances de ces logiciels a été proposée par Kong *et al.* [Kon 96]. Lors de la session suivante, GREC'97, un tournoi a porté sur les logiciels de vectorisation. Un rapport sur ce tournoi peut-être consulté à [Chh 98b]. Plus récemment, Phillips *et al.* ont proposé un protocole pour l'évaluation de performances des systèmes de reconnaissance de graphiques [Phi 98]. De leur côté, Wenyin et Dori ont proposé une évaluation de performances de la séparation texte / graphique [Wen 98a]. Lors de GREC'99, Chhabra et Phillips ont proposé une mesure de performances basée sur la définition d'une fonction de coût [Chh 99].

Pour intégrer plus fortement l'évaluation de performances, Smeulders et de Boer proposent plus qu'une simple méthode [Sme 98]. Dans leurs travaux, ils décrivent une architecture logicielle incluant des aspects d'évaluation, constituée de deux systèmes : un adapté à leur problème (la conversion de documents techniques) et un autre conçu à des fins d'évaluation de performances. Ce second système est un outil comprenant une base de données d'exemples complétée au fur et à mesure des tests. Il est séparé du premier système et contient trois composantes : une première pour stocker des données, une seconde dédiée à l'exécution des expérimentations et une troisième chargée de l'analyse des données. Le premier système ne constitue qu'un paramètre du second, qui peut donc s'adapter à d'autres systèmes. L'ensemble est très modulaire, l'architecture logicielle proposée peut être utilisée pour d'autres travaux, ce qui constitue un atout supplémentaire.

Plusieurs conclusions peuvent être tirées de cette activité. D'une part, le nombre de travaux liés à l'évaluation est bien représentatif de l'intérêt porté par les scientifiques de notre domaine à ce principe. D'autre part, les évaluations essentiellement privilégiées portent sur le codage, ce type d'évaluation étant plus simple à réaliser. L'évaluation reste cependant parfois limitée. En particulier, un des problèmes soulevés par ces évaluations est l'absence d'une base de données de référence, permettant à différents chercheurs du domaine de tester leurs traitements à partir de données communes. Si des efforts ont déjà été réalisés dans ce sens pour des documents de type bureautique [Phi 93], il reste à établir une base de données plus variée regroupant différents types de documents techniques. Après une période assez importante de développement relative à notre environnement d'analyse de plans, nous sommes tournés actuellement vers cette orientation et souhaitons contribuer à cet effort communautaire.

2.4 Principes retenus et guide de lecture

Nous nous proposons d'introduire les concepts définis dans cette thèse selon la terminologie présentée ci-dessus. Nous adoptons ainsi, autant que possible, une structuration de nos idées autour du schéma *Méthode–Algorithme–Mise en œuvre*. En particulier, les parties 2, 3, et 4 présentent nos travaux sur l'analyse de plans architecturaux, où sont détaillées les méthodes que nous utilisons. Nous introduisons dans ces parties certains des algorithmes issus des méthodes développées, afin de présenter nos choix de conception à ce niveau. La mise en œuvre de ces traitements est parfois évoquée par quelques données qualitatives et quantitatives.

La partie 5 est entièrement dédiée à la mise en œuvre générale du système et s'attarde sur les choix d'architecture logicielle que nous avons adoptés. Dans cette organisation, l'évaluation de performances est parfois évoquée. Cet aspect n'est cependant pas très développé, même s'il a été pris en compte pour la

conception globale du système, et reste à approfondir. Nous fournissons à ce titre quelques perspectives envisageables à la fin de ce mémoire.

Deuxième partie

Bas-niveau

Chapitre 3

Traitements appliqués aux images numérisées

3.1 Introduction — Acquisition

Avant l'avènement de l'informatique, une grande partie des documentations était éditée sur support papier. Afin de faciliter le stockage et l'indexation de ces documentations, des projets d'analyse de documents ont commencé à émerger. Ces analyses de documents se décomposent en plusieurs étapes, dont la première est la numérisation du document. Cette étape consiste à acquérir une source d'information extérieure à l'ordinateur — dans notre cas le document — par l'intermédiaire d'un périphérique jouant le rôle de capteur. De nombreux types de capteurs existent, dotés de caractéristiques qui les prédisposent à certains domaines d'utilisation. Parmi les capteurs utilisés en analyse de documents, citons les caméras numériques ou à capteurs CCD [11], les scanners à main ou à plat.

En analyse de documents techniques, il est généralement admis qu'une résolution variant de 300 à 600 dpi [12] est nécessaire afin de disposer de données assez denses. Ce type de résolution n'est pour le moment disponible qu'à partir de scanners, qui sont ainsi en règle générale les périphériques d'acquisition privilégiés pour ce domaine d'application. Nous utilisons donc ce type de matériel pour nos besoins. Cela présente un avantage supplémentaire : la possibilité de numériser de grands documents en une seule fois ; il existe en effet des scanners capables de traiter des documents allant jusqu'au format A0. La caméra est cependant utilisée dans certains domaines proches du notre. C'est le cas par exemple pour Zappalá *et al.* [Zap 99] qui l'utilisent pour numériser des journaux, destinés à être analysés par des OCR.

À l'issue de cette phase d'acquisition, on obtient une image représentant le document étudié. Cette image, appelée *image bitmap* ou *image numérisée*, correspond à une matrice de points, dont la taille est proportionnelle à la taille du document et à la résolution utilisée. Chaque point de l'image est attribué par

11. *Charge-Coupled Device.*
12. *Dots per inch* : nombre de points par pouce.

une couleur, déterminée par le signal lumineux perçu par le capteur au niveau de la zone correspondante du document. C'est à partir de cette image numérisée que l'analyse du document commence. Ce chapitre a pour but de décrire les différents traitements de l'analyse qui sont appliqués sur les images numérisées : les binarisations [§ 3.2], les segmentations [§ 3.3] et l'exploitation des images contenant du texte [§ 3.4].

3.2 Binarisation

3.2.1 Introduction

L'information représentée dans les documents techniques que nous étudions (cadastre, plans mécaniques ou architecturaux) est constituée d'éléments graphiques et textuels. Ces éléments correspondent *a priori* à une information binaire. En effet, les nuances de couleur — ce qui inclut également les nuances de gris — sont très rarement utilisées pour apporter des informations sémantiques au contenu du document. Cependant, même si l'information contenue dans un document est de nature binaire, les données extraites à l'issue de la numérisation présentent une grande variété de nuances, et pas seulement deux nuances idéales. Ce phénomène peut résulter de plusieurs facteurs, comme une conception non uniforme, une réflectance non homogène ou encore des transitions entre les contours des régions représentant l'information et le fond du document [O'G 95]. De plus, ces facteurs peuvent être amplifiés si la qualité générale du document initial est mauvaise ou s'il contient du bruit, telles que des traces résultant d'une pliure, d'un déchirement, d'une salissure, etc.

L'objectif de la *binarisation* est d'identifier les points qui appartiennent à l'information véhiculée. Ces points sont en principe décrits par une seule nuance de couleur, par différence avec les points appartenant au fond, décrits par les autres intensités. Nous décrivons dans ce paragraphe les méthodes de binarisation que nous utilisons pour traiter nos documents. Pour un état de l'art complet en binarisation, le lecteur pourra se reporter à [Wes 78, Tri 95a, Tri 95b].

3.2.2 Binarisations globales

Pour les documents « propres » et présentant un contraste convenable entre les données et le fond du document, il est possible de procéder à une *binarisation globale* du document. Le principe est assez simple : après avoir déterminé une nuance seuil, tous les points de l'image numérisée sont parcourus. Ceux dont la nuance est plus claire que la nuance seuil sont identifiés comme appartenant au fond du document, les autres comme appartenant à l'information véhiculée par le document. Pour déterminer la nuance seuil, on effectue généralement une étude de l'histogramme des couleurs de l'image. Comme l'information véhiculée par cette image est de nature binaire, l'histogramme obtenu est un histogramme bimodal, présentant deux pics distincts. La valeur seuil est alors calculée comme la nuance séparant le mieux les deux pics. Une méthode simple d'effectuer ce calcul est de rechercher le minimum de l'histogramme entre ses deux maximums. D'autres méthodes plus sophistiquées ont également été proposées [O'G 95].

Ce type de binarisation est utilisée par les scanners pour produire des images binaires. Ils considèrent une intensité lumineuse I_v faisant office de seuil : les zones du document dont l'intensité est inférieure à I_v sont considérées comme noires, et celles supérieures à I_v comme blanches. Cette technique permet de fournir de bons résultat sur des documents relativement propres, mais est cependant limitée sur les documents plus bruités. En effet, si le seuil I_v est mal déterminé par le scanner, on obtient des images dont une partie de l'information est manquante, ou des images dont une partie du fond est reconnue

comme de l'information [FIG. 3.1(b), 3.1(c)]. L'utilisateur n'ayant généralement pas la maîtrise du seuil I_v, ce type d'acquisition s'avère alors inadapté.

D'autres possibilités de numérisation sont alors possibles, comme la numérisation en niveaux de gris. Chaque point de l'image produite est alors attribué par une nuance de gris, parmi les 256 disponibles dans les palettes de couleurs standards. À partir de cette image, il nous est possible d'appliquer une binarisation à base de seuil fixe, semblable à celle proposée par les scanners, tout en gardant la maîtrise du seuil I_v. Ce seuil peut alors être déterminé soit manuellement, soit au moyen d'un calcul préliminaire portant sur l'étude de l'histogramme des niveaux de gris de l'image, ou, de façon plus générale, sur l'optimisation d'un critère global défini sur l'image.

3.2.3 Binarisations locales

Dans certaines situations, une binarisation globale ne peut pas apporter de résultat satisfaisant. C'est notamment le cas lorsque l'image initiale est extrêmement bruitée, pour une des raisons évoquées en introduction, ce qui peut produire des images binaires cumulant les deux types de défaut énoncés pour le cas de la binarisation globale par scanner. Cette situation nous a amené à nous tourner vers d'autres types de méthodes de binarisation, basées sur une étude des propriétés locales de l'image. Ces méthodes sont qualifiées de *binarisations locales* ou encore de *binarisation adaptatives*. C'est le cas de la méthode proposée par Trier et Taxt [Tri 95c], basée sur un seuillage local adaptatif. Cette méthode, elle-même inspirée de la méthode de White et Rohrer [Whi 83], se décompose en plusieurs étapes :

1. calcul du gradient de l'image d'origine lissée, par convolution de l'image initiale par la dérivée d'une gaussienne [Can 86] ;
2. calcul de l'image G du module de ce gradient :
$$G = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$
3. calcul de l'activité $A(x,y)$ de l'image, suivant la formule :
$$A(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} G(x+i, y+j)$$
4. calcul du laplacien $\nabla^2 I$ de l'image par convolution avec le laplacien d'une gaussienne ;
5. création d'une image où chaque pixel vaut 0, + ou -, avec un seuil d'activité T_A, selon l'algorithme suivant :
$$L(x,y) = \begin{cases} \text{'0'} & \text{si} \quad A(x,y) < T_A \\ \text{'-'} & \text{si} \quad A(x,y) \geqslant T_A \quad \text{et} \quad \nabla^2 I(x,y) < 0 \\ \text{'+'} & \text{si} \quad A(x,y) \geqslant T_A \quad \text{et} \quad \nabla^2 I(x,y) \geqslant 0 \end{cases}$$
6. pour chaque composante 4-connexe r de valeur '0' dans L, calcul du nombre N^- de pixels '-' et du nombre N^+ de pixels '+' qui sont 8-connexes avec r. Si $N^+ > N^-$, étiquetage de tous les pixels de r par '+' ;
7. seuillage de l'image L en gardant tous les pixels marqués '+' pour l'information, les autres appartenant au fond ;
8. pour toutes les composantes 4-connexes de l'image binaire ainsi obtenue, calcul de la moyenne du module du gradient à la périphérie de la composante, et suppression des composantes pour lesquelles cette moyenne est inférieure à un seuil T_P.

Nous utilisons cette méthode pour les documents bruités, notamment ceux comportant des traces de pliures. La méthode produit d'assez bons résultats [FIG. 3.1(d)], mais a un inconvénient majeur : il n'existe pas d'interprétation des paramètres T_A et T_P. Les valeurs sont ainsi difficiles à déterminer, et restent fixées expérimentalement pour l'instant.

3.2.4 Perspectives

Nous nous orientons actuellement vers d'autres pistes pour disposer d'une binarisation robuste, en tentant de limiter le nombre de paramètres nécessaires ou de diminuer la complexité de leur calcul :

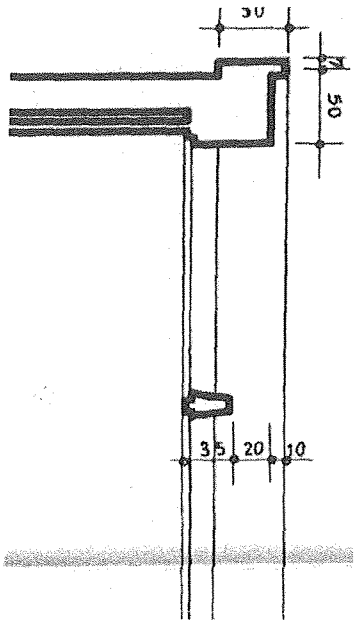
- la binarisation floue, à partir de la méthode proposée par Cheng et Chen [Che 99]. Ils proposent d'effectuer un partitionnement flou de l'image traitée, à partir duquel ils estiment localement l'entropie maximum au moyen d'un algorithme génétique. Cela leur permet de déduire les valeurs d'un masque à partir duquel l'image est seuillée. Les résultats obtenus sont intéressants et la méthode a l'avantage de ne pas nécessiter de paramètre ;
- une méthode de binarisation par seuillage automatique, développée dans l'équipe [Tab 00b]. Elle est basée sur un traitement itératif de l'image, segmentée en régions par application d'un opérateur Laplacien, qui consiste à appliquer un test statistique d'homogénéité à chaque région. Ce test classifie itérativement les régions traitées jusqu'à l'obtention de l'image binaire finale. Les premières expérimentations de cette méthode montrent de bons résultats, même si des limites sont imputables à l'utilisation du Laplacien, typiquement sur des profils de type « escalier ».

3.3 Segmentations

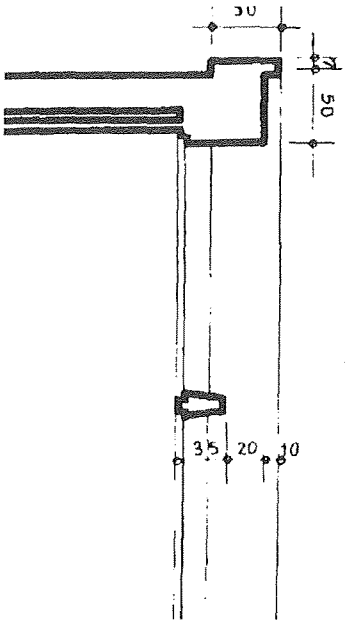
3.3.1 Introduction

Une fois l'image binaire obtenue, l'information pertinente contenue correspond aux pixels noirs de cette image. On peut considérer que cette information correspond à la superposition d'un certain nombre de calques, chacun des calques contenant une partie de l'information globale. Certains types de documents se prêtent particulièrement bien à cette décomposition en calques, ou *couches logiques*. C'est le cas par exemple des cartes géographiques composées d'une couche de relief, d'une couche d'hydrométrie, d'une couche correspondant au réseau routier, d'une couche toponymique, etc. Les traitements permettant d'extraire une couche entrant dans la composition d'un document peuvent être regroupés sous le terme générique de *segmentations*. Au niveau de l'image, il est cependant parfois difficile de séparer les différentes couches logiques lorsque cela nécessite de disposer de connaissances sur la structure de l'image, qui ne sont généralement pas disponibles à cet état d'avancement de l'analyse. Il est par contre plus simple de s'intéresser à une décomposition en *couches physiques*, qui correspondent à des critères mathématiques. Les segmentations peuvent alors être décrites par des méthodes basées sur l'étude de ces critères mathématiques.

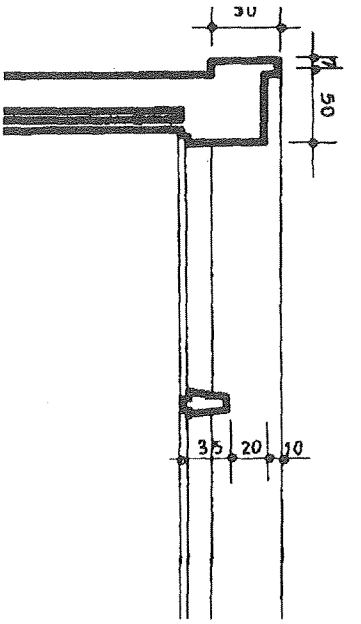
Les segmentations sont très intéressantes : lorsque l'information qui correspond à une couche est bien isolée du reste de l'image, l'analyse de cette couche est grandement facilitée. En effet, les traitements dédiés au type d'information présente sur la couche peuvent être appliqués avec un risque moindre de bruit que lorsque cette information est confondue, sur le document initial, avec l'information véhiculée par les autres couches. Nous présentons ci-dessous diverses segmentations applicables sur les plans d'architecte, en nous restreignant aux segmentations qui sont applicables sur les images de points. Ces segmentations



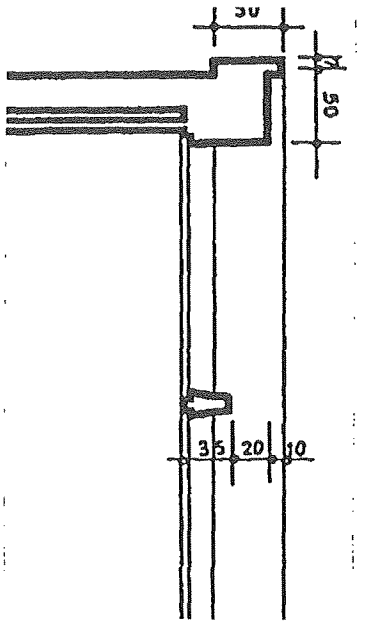
(a) Image initiale.



(b) Binarisation fixe, $I_v = 90$.



(c) Binarisation fixe, $I_v = 130$.



(d) Binarisation adaptative, $\sigma = 2$, $T_A = T_P = 6$.

FIG. 3.1 – *Les images binaires obtenues à partir d'une image en niveaux de gris bruitée (a). La binarisation fixe sur ce type d'image peut provoquer, suivant la valeur de seuil utilisée, une perte d'information (b) ou du bruit sur l'information extraite (c) — notamment au niveau des traits gras en haut à gauche qui « bavent ». La binarisation adaptative permet sur ce type d'image d'extraire des images claires (d), en dehors des artefacts produits sur les bords de l'image.*

permettent de répartir les pixels noirs — représentant l'information — présents sur l'image binaire initiale vers plusieurs images binaires, chacune d'elles correspondant à une couche. Ces traitements ne sont pas automatiquement inclus dans la chaîne des traitements que nous appliquons sur un document. Ils sont utilisés en fonction des besoins et du type de plan que nous analysons.

3.3.2 Séparation texte / graphique

Les plans d'architecte permettent de décrire la structure d'un bâtiment à partir des différents éléments architecturaux qui les composent, en termes de murs, portes, fenêtres, cloisons, etc. Ces éléments sont représentés par des symboles décrits sous forme graphique, généralement à partir de segments et d'arcs de cercle, attribués par une épaisseur. Il n'existe pas de standard de représentation de ces symboles, et on trouve ainsi des conventions de représentation différentes pour chaque architecte, quand ce n'est pas pour chaque plan. Ces informations peuvent être qualifiées d'informations purement graphiques, par opposition aux informations textuelles apportant des renseignements sur la fonction attribuée aux pièces (cuisine, entrée, ...), à la valeur des différentes cotations, etc. Il y a donc un intérêt à segmenter ces deux couches logiques, afin de leur appliquer par la suite les traitements qui les concernent.

Cette segmentation logique, difficile à réaliser, peut être approchée par une segmentation physique. Ce type de segmentation est un problème bien connu en analyse de documents et a généré une littérature abondante. Le lecteur pourra se reporter à [Doe 98] pour un état de l'art des méthodes développées. Ces méthodes sont actuellement *a priori* matures, au moins lorsque le texte ne touche pas la partie graphique du document ; le problème reste ouvert dans ce cas [Tom 98b].

Pour notre part, nous avons choisi une approche basée sur l'étude des composantes connexes, considérée comme robuste. Dans un premier temps, nous considérons comme texte et assimilés les petites composantes connexes de l'image. Nous avons choisi une implantation de la méthode de Fletcher et Kasturi [Fle 88], en adaptant ses paramètres au type de documents traités dans cette application et en ajoutant un seuil absolu sur la longueur et la largeur d'une composante de type texte. La méthode se base sur une étude des dimensions des composantes connexes, c'est-à-dire des groupes de pixels noirs contigus. L'algorithme ainsi modifié procède comme suit :

1. calcul de l'histogramme des aires des composantes connexes ;
2. détermination de la zone la plus peuplée de cet histogramme (A_{pp}) ;
3. calcul de la moyenne de l'histogramme (A_{moy}) ;
4. détermination d'un seuil de taille S_1 à $3 \times \max(A_{pp}, A_{moy})$;
5. détermination d'un rapport maximal d'élongation S_2 — dans nos expérimentations, la valeur usuelle de ce rapport est de 20 ;
6. détermination d'un seuil absolu de longueur et de largeur S_3 ;
7. filtrage de l'ensemble des composantes connexes en étiquetant comme texte celles dont l'aire est inférieure à S_1, dont le ratio hauteur / largeur est compris entre $1/S_2$ et S_2, et dont la hauteur et la largeur sont toutes deux inférieures à S_3.

Nos expérimentations indiquent une bonne stabilité de ces seuils, pour une échelle de résolution donnée. Au niveau des traits tiretés, nos premières expérimentations avec des méthodes effectuant la détection de ces primitives sur l'image de points, telle que celle proposée par Agam *et al.* [Aga 96], ont été assez décevantes. Ceci nous a amené à nous tourner vers des détections de traits tiretés à partir de données vectorielles (voir chapitre 6). Notre seule préoccupation à ce stade de l'analyse est donc de segmenter les tiretés correspondant aux traits tiretés sur la couche graphique, afin de pouvoir les exploiter

ultérieurement. Cependant, à l'issue de l'algorithme exposé ci-dessus, c'est sur la couche de texte que sont déplacés *tous* les tiretés, car ils respectent les contraintes de segmentation de texte énoncées ci-dessus. Nous avons donc défini un post-traitement sur les composantes connexes segmentées vers la couche de texte, afin de réintégrer tous les tiretés sur la couche graphique :

1. calcul du rectangle englobant[13] d'aire minimale (REAM) de la composante connexe selon la méthode de Freeman et Shapira [Fre 75] ;
2. détermination d'un seuil S_4 de densité minimum de pixels noirs de la composante connexe, calculé comme le rapport du nombre de pixels noirs de la composante sur l'aire de son REAM. Cette contrainte permet de ne retenir que les composantes denses en pixels noirs dont la forme est proche de celle d'un rectangle ;
3. détermination d'un rapport minimal d'élongation S_5 ;
4. déplacement vers la couche graphique des composantes connexes dont la densité minimum de pixels noirs est supérieure à S_4 et dont le ratio hauteur / largeur n'est pas compris entre $1/S_5$ et S_5.

Des résultats expérimentaux obtenus par cette méthode sont présentés [FIG. 3.2]. On constate que plusieurs problèmes subsistent :

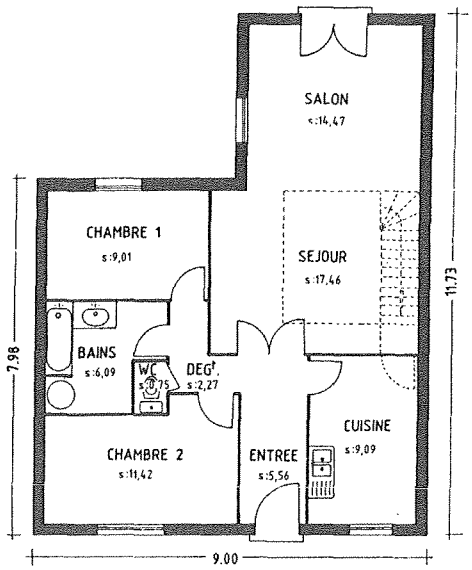
- si la majeure partie des traits tiretés se retrouvent sur la couche graphique, les traits tiretés qui s'intersectent restent sur la couche texte. Il est en effet difficile de les séparer des caractères sans connaissance supplémentaire *a priori* ;
- les lettres « I », assimilables à des tirets, c'est-à-dire à des tiretés typographiques, sont mal segmentées et se retrouvent ainsi sur la couche graphique ;
- c'est aussi le cas des caractères connexes à la couche graphique qui ne sont pas segmentés sur la couche texte. Si des propositions ont été faites pour résoudre ce type de problème [Luo 94, Dor 96a, Tof 98], l'ajustage de ces post-traitements reste très délicat. En effet, les post-traitements permettant de rectifier certaines erreurs en introduisent également de nouvelles. Les réponses restent donc partielles et nous avons décidé de suivre une autre approche pour résoudre nos problèmes de segmentation.

En effet, étant donné que tous nos traitements sont intégrés au sein d'une interface homme-machine qui offre des possibilités d'interaction, nous avons intégré à l'interface la possibilité de corriger ces erreurs de segmentations de façon manuelle [§ 15.6.2]. C'est un compromis *a priori* acceptable, au vu de la complexité du problème et des contraintes liées à l'aboutissement d'un environnement de suivi d'analyse de plans architecturaux. En outre, le temps nécessaire à la correction manuelle est lié au pourcentage d'erreurs de segmentation, qui est généralement raisonnable. La figure [FIG. 3.3] présente les couches finales obtenues après rectification sous l'interface MICA (présentée plus en détail dans le chapitre 15).

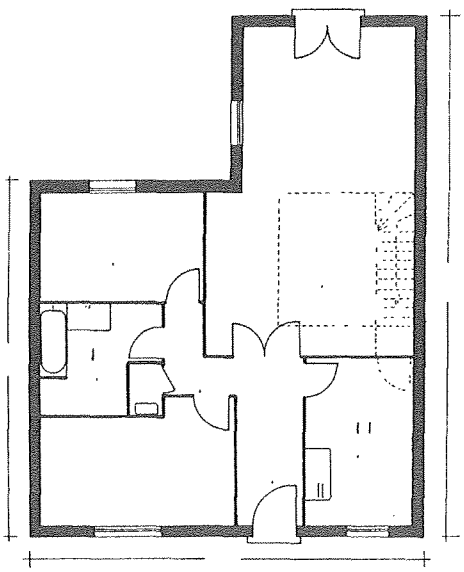
3.3.3 Séparation traits forts / traits fins

La couche graphique extraite est la couche décrivant les différents éléments architecturaux qui composent le plan. Ces éléments sont représentés par des symboles décomposables en primitives graphiques : des segments et des arcs de cercle. Si certains symboles architecturaux sont composés de plusieurs de ces primitives, ils en existent également, suivant les plans, qui correspondent directement aux primitives elles-mêmes. Typiquement, les murs sont souvent simplement représentés par des segments. C'est le cas des murs porteurs et des cloisons présents dans les plans de pavillon étudiés ci-avant. L'épaisseur

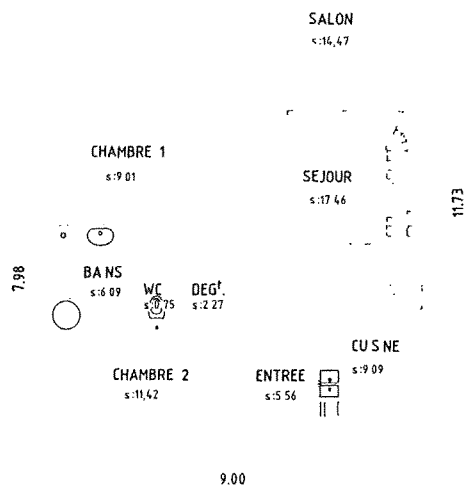
13. La nuance faite ici entre les rectangles et les boîtes englobantes tient au fait que les côtés des boîtes englobantes sont parallèles aux axes, ce qui n'est pas obligatoire pour les côtés des rectangles englobants.



(a) Image originale.

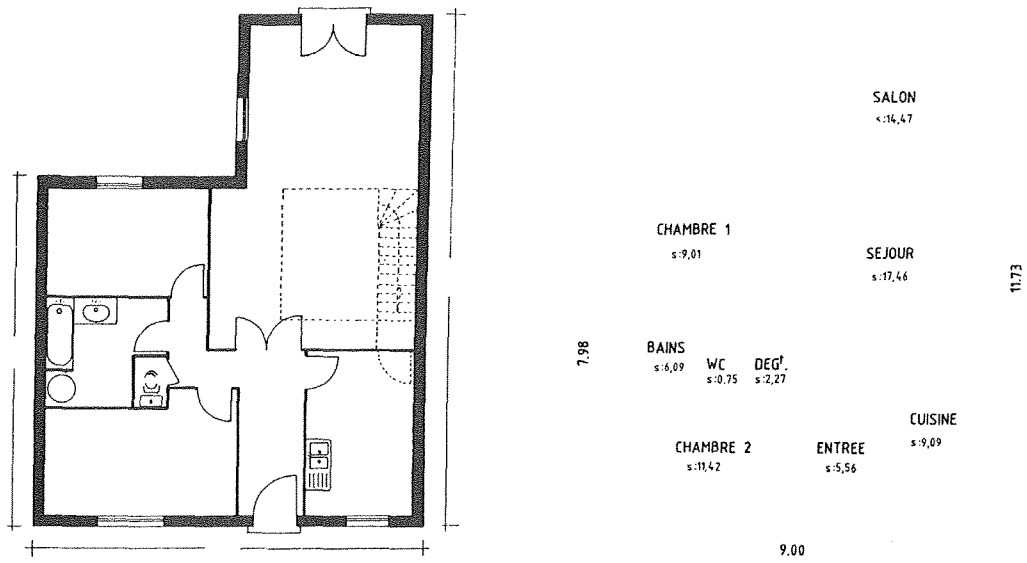


(b) Couche graphique extraite.



(c) Couche texte extraite.

FIG. 3.2 – *Exemple de séparation texte / graphique sur les plans d'un pavillon.*



(a) Couche graphique finale.

(b) Couche texte finale.

FIG. 3.3 – *Couches corrigées correspondant aux couches de la figure 3.2.*

des segments est alors utilisée afin de différencier ces deux symboles architecturaux. Cet exemple illustre l'intérêt d'un traitement permettant d'effectuer une segmentation des traits forts et des traits fins. Dans ce cas précis, cela permet d'obtenir directement une couche contenant tous les murs porteurs d'un pavillon. Cela peut également permettre d'isoler d'autres types d'information sur des plans ayant des conventions différentes de représentation.

Afin de nous doter d'un traitement permettant d'effectuer une segmentation traits forts / traits fins, nous avons choisi d'utiliser des opérateurs de morphologie mathématique [Ser 82]. Ces opérateurs sont assez robustes et permettent d'obtenir des résultats de grande qualité. L'algorithme de segmentation traits forts / traits fins, pour une image I donnée et pour une épaisseur seuil e, se décompose de la manière suivante :

1. à partir du calcul de $n = \lfloor \frac{e}{2} \rfloor$ (où la notation $\lfloor \rfloor$ correspond à la fonction d'arrondi inférieur), utilisation d'un élément structurant B_n, carré de côté $2n + 1$, pour calculer l'image *érodée* J correspondant à l'image I :

$$J = I \ominus B_n.$$

   Dans cette phase, les lignes présentes dans l'image I sont amincies d'une épaisseur de $2n$ pixels, ce qui provoque la suppression des lignes dont l'épaisseur est inférieure à $2n$ pixels dans J ;
2. reconstitution des traits forts par *reconstruction géodésique* partielle ($n + 1$ itérations) :

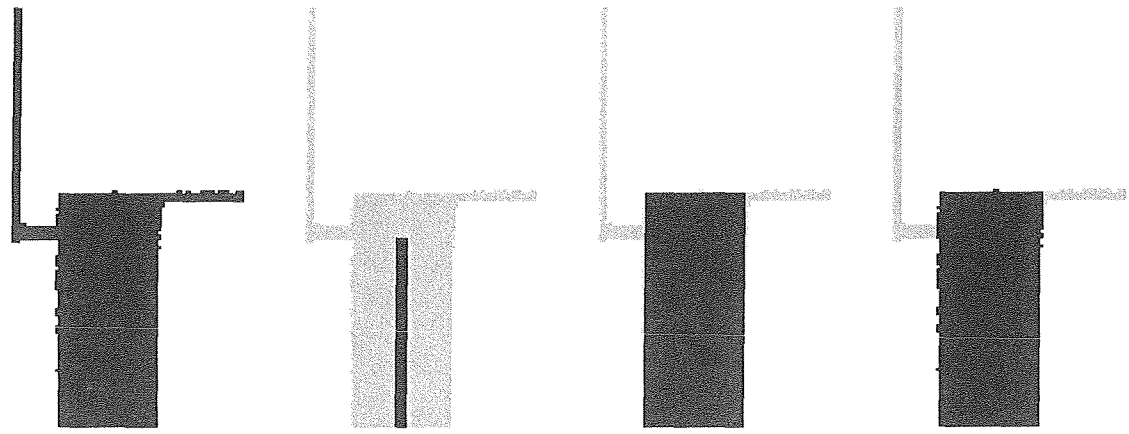
$$\begin{aligned} K_0 &= J, \\ K_i &= (K_{i-1} \oplus B_1) \cap I \quad \text{pour} \quad i = 1 \ldots n+1. \end{aligned}$$

   Cette reconstruction géodésique permet de restituer les traits forts tels qu'ils existaient initialement, par comparaison avec l'image originale. Ce n'est pas le cas lors de l'utilisation d'une *dilata-*

tion, qui ne permet en effet que d'obtenir une approximation du résultat obtenu par reconstruction géodésique ;

3. calcul des deux images $I_{fort} = K_{n+1}$ et $I_{fin} = I - I_{fort}$.

La figure [FIG. 3.4] permet de suivre sur un exemple le résultat engendré par les divers opérateurs morphologiques présentés. La figure [FIG. 3.5] illustre le résultat de la séparation traits forts / traits fins sur la couche graphique d'une image binaire. Les résultats obtenus grâce à ce traitement sont relative-



(a) Image originale. (b) Érosion. (c) Dilatation. (d) Reconstruction géodésique.

FIG. 3.4 – *Présentation sur un exemple des différents opérateurs morphologiques utilisables pour la segmentation traits forts / traits fins [AS 98b].*

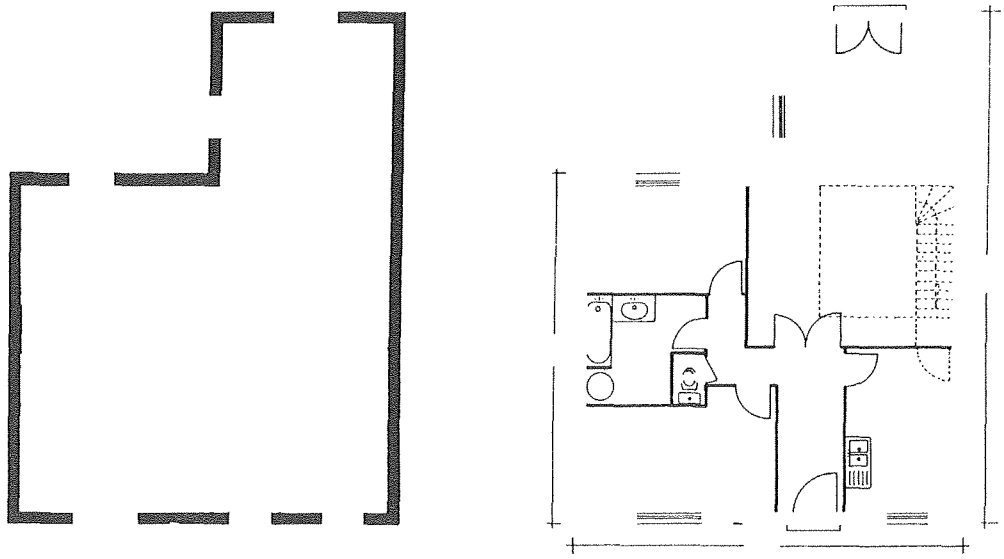
ment propres. Il est conseillé toutefois de ne pas l'exécuter plusieurs fois sur une image donnée, pour obtenir une segmentation en plus de deux couches. En effet, la continuité des traits segmentés n'est pas assurée par ce type de méthode, ce qui peut provoquer des pertes d'informations par rapport à l'image initiale.

3.4 Exploitation de la couche texte

Les caractères présents sur la couche texte peuvent être exploités : regroupés en chaînes, par la suite analysées par un OCR [14], ils peuvent permettre d'identifier les différentes pièces d'un plan, de connaître la valeur des cotations, etc. Pour regrouper les caractères en chaînes de caractères, nous avons implanté la méthode de Fletcher et Kasturi [Fle 88], qui est à base de transformée de Hough [Pra 78], permettant de trouver les alignements des caractères. Les principales phases de l'algorithme sont :

1. recherche des alignements horizontaux et verticaux des composantes connexes (des caractères) grâce à la transformée de Hough ;
2. recherche, si nécessaire, des alignements diagonaux sur les composantes non encore regroupées, toujours grâce à la transformée de Hough ;

14. *Optical Character Recognition.*



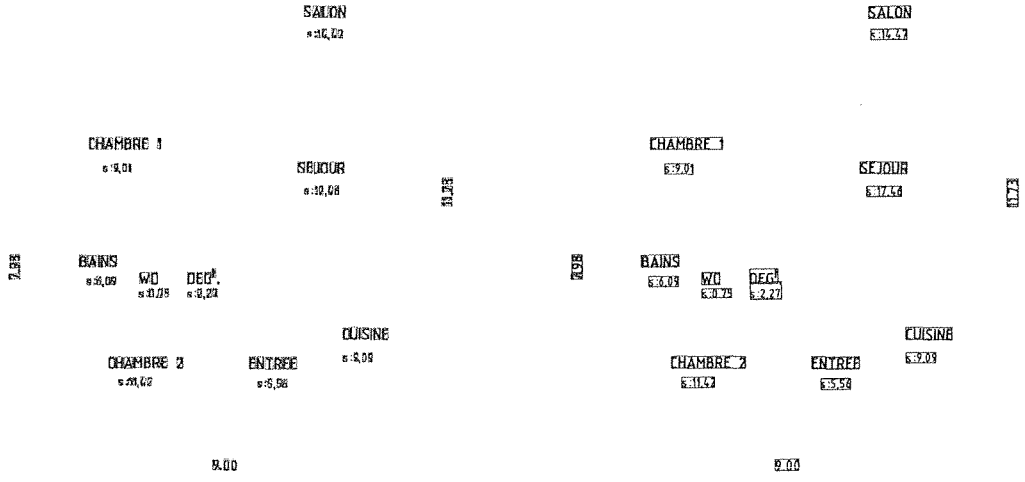
(a) Traits forts extraits. (b) Traits fins extraits.

FIG. 3.5 – *Segmentation traits forts / traits fins réalisée à partir de la figure 3.3(a).*

3. pour chaque alignement trouvé, regroupement des caractères en mots, en tenant compte de seuils de distance entre composantes et d'autres critères géométriques, comme la hauteur moyenne des caractères de l'alignement.

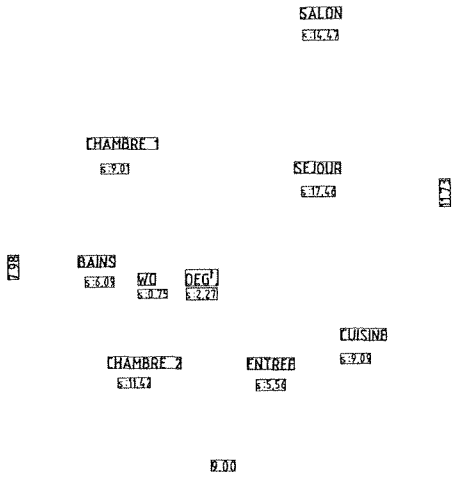
Les résultats obtenus par cette méthode sont de bonne qualité [FIG. 3.6], même s'ils présentent quelques défauts. Il existe plusieurs façons d'améliorer la qualité des résultats obtenus. Cette méthode peut par exemple être complétée par l'emploi de règles de regroupement structurel similaires à celles qui ont été élaborées dans un travail antérieur sur l'analyse de la cotation en dessin technique [Col 92]. Dans un premier temps cependant, le nombre de corrections à effectuer étant limité, nous avons doté l'interface homme-machine supervisant l'analyse d'outils supplémentaires pour pouvoir effectuer ces corrections manuellement [§ 15.6]. L'utilisateur a ainsi la possibilité depuis l'interface de valider ou de corriger les résultats issus de cette phase automatique, par la suppression, la création ou la modification des boîtes englobantes. Cette phase de correction manuelle permet d'obtenir des résultats très satisfaisants [FIG. 3.6]. Comme dans le cas de la séparation texte / graphique, cette solution paraît représenter un bon compromis par rapport à l'ajout d'un post-traitement, qui ne serait peut-être pas non plus exempt d'artefacts.

La localisation des chaînes de caractères peut dès lors constituer le paramètre d'un logiciel de reconnaissance de caractères, afin d'en faciliter la tâche.



(a) Caractères localisés.

(b) Chaînes localisées.



(c) Chaînes localisées après correction.

FIG. 3.6 – *Localisation des chaînes de caractères à partir de la couche texte de la figure 3.3(b). Deux corrections manuelles ont été apportées : la fusion des boîtes englobantes de la cotation la plus à droite en une seule, et le redimensionnement des boîtes englobantes du texte lié au couloir (noté « Deg^{t} » sur le plan), le point final du libellé ayant été incorporé dans la chaîne correspondant à la surface.*

Chapitre 4

Tuilage

4.1 Introduction

Dans ce mémoire, nous mettons en évidence deux grands cycles de traitements intervenant dans le processus global d'analyse de plans architecturaux. Le premier, qui est souvent appelé *bas-niveau*, correspond à l'ensemble des traitements appliqués aux images numérisées. Il recouvre dans notre cas les étapes allant de la binarisation à la vectorisation. Le deuxième, appelé *haut-niveau*, correspond à l'ensemble des traitements qui suivent la vectorisation. Ces traitements s'appliquent à des données plus symboliques — primitives graphiques, primitives architecturales — ce qui leur confère ce caractère plus « haut-niveau ».

Dans notre contexte, la manipulation des images lors des opérations bas-niveau peut nécessiter une quantité de mémoire conséquente qui n'est pas toujours disponible, même sur une station de travail haut de gamme. En effet, il n'est pas rare de disposer de plans architecturaux au format A0, représentant approximativement 130 Mo lorsqu'ils sont numérisés à 300 dpi en 256 niveaux de gris, la taille de l'image étant directement proportionnelle à la taille du plan de départ. Lors de l'application de masques de convolutions sur de telles images, la ressource en mémoire nécessaire peut être 8 fois supérieure à la taille originale [§ 3.2], ce qui représente environ 1 Go. De telles quantités de mémoire sont encore très importantes, même en bénéficiant des techniques d'adressage virtuel disponibles sur la plupart des systèmes d'exploitation.

Dans la suite de notre analyse, à partir des opérations haut-niveau, les images sont décrites par des ensembles de primitives graphiques. La place mémoire occupée dépend alors du nombre de ces primitives et non plus de la taille du plan original. Généralement, les images comportent alors quelques centaines, voire quelques milliers de primitives, ce qui nécessite relativement peu de ressources mémoire comparativement aux dizaines de millions de pixels nécessaires pour décrire l'image de points corres-

pondante. Dans ces conditions, les images peuvent alors être manipulées dans leur globalité. Le seul passage critique au niveau des ressources mémoire se situe donc lors des traitements bas-niveau.

La première solution pour contourner ce problème est d'utiliser des utilitaires ou des bibliothèques conçus pour manipuler de grandes images. Cela permet de travailler de façon transparente sur une grande image stockée sur disque comme si cette image était présente entièrement en mémoire centrale. La partie de l'image effectivement présente en mémoire ne représente alors qu'une fenêtre de l'image du disque, la mise à jour de cette fenêtre au cours des besoins étant effectuée automatiquement. De tels produits commerciaux existent, ainsi que des produits libres comme LIMP [15], bibliothèque C++ sous licence LGPL [16] permettant d'intégrer des traitements sur les images manipulées [Gou 00].

L'autre solution permettant de résoudre le problème de la saturation mémoire est de gérer directement la manipulation de grandes images. C'est cette solution qui est décrite dans ce chapitre [Dos 00a].

4.2 Quelques travaux similaires

Afin de permettre la manipulation même des informations, il est nécessaire de décomposer l'image initiale. Une des approches envisageables consiste à découper l'image initiale en un certain nombre d'images plus petites, dont la taille autorise un traitement individuel. À l'issue de ces traitements, ces images, appelées *tuiles*, sont fusionnées pour reconstituer une image résultat complète.

Janssen et Vossepoel [Jan 94] adoptent cette démarche pour vectoriser des dessins de dimensions trop importantes pour être numérisés d'un seul tenant. Ils effectuent des numérisations de plusieurs parties du même dessin, de façon à en couvrir l'ensemble, et vectorisent séparément chacune des tuiles. Afin de reconstituer le dessin global, ils déterminent des points de contrôle à partir des données vectorisées de leurs tuiles. Un appariement de ces points de contrôle leur permet de calculer la transformation géométrique qui permet de recaler les tuiles les unes par rapport aux autres.

Vossepoel *et al.* [Vos 97] adoptent la même démarche pour permettre la squelettisation de grands documents techniques. Ceux-ci sont découpés en un certain nombre de tuiles de taille plus raisonnable, qui sont ensuite traitées indépendamment les unes des autres. À l'issue de la squelettisation, les tuiles sont fusionnées pour constituer le document final global. À cette fin, une zone de recouvrement est définie entre chaque paire de tuiles voisines pour faciliter la mise en correspondance des morceaux de squelette qu'elles contiennent. Le rôle de la zone de recouvrement est primordial : sa taille doit être calculée au plus juste afin de ne perdre aucune information capitale pour la fusion, tout en réduisant au minimum le surplus de données à traiter engendré par l'utilisation de cette zone.

Cette zone de recouvrement est utilisée dans d'autres contextes assez similaires. Dans [Whi 98], Whichello et Yan exposent la démarche qu'ils adoptent pour traiter une page de journal, destinée à être analysée par un système de reconnaissance de caractères, lorsqu'il est impossible techniquement de numériser cette page d'un seul tenant. Ils la numérisent alors en plusieurs portions couvrant l'ensemble de la page et comportant une zone de recouvrement entre elles. À partir de ces zones de recouvrement, ils effectuent des recalages entre les différentes portions, ce qui leur permet de reconstituer la page de journal dans sa globalité.

Zappalá *et al.* [Zap 99] rencontrent également des problèmes relatifs à la taille des données lors de la numérisation d'un document avec une caméra vidéo qui fait office de scanner. La caméra présente des avantages par rapport à un scanner « classique », comme la commodité de manipulation du document,

15. *The Large Image Manipulation Program.*
16. *GNU Lesser General Public License.*

mais a aussi ses défauts, la résolution maximum ne pouvant atteindre que 50 dpi sur l'acquisition complète d'une feuille A4. Cette résolution n'étant pas suffisante pour une analyse par OCR, une mosaïque d'images, plus petites et ainsi plus précises, est obtenue par numérisation du document original. Les images extraites comportent une zone de recouvrement avoisinant 50 %, ce qui représente dans leur cas un bon compromis entre la précision des données obtenues et l'efficacité de leur traitement. Le recalage des images s'effectue après mise en correspondance d'indices extraits à partir des différentes images et permet de reconstituer le document original.

4.3 Principe du tuilage

Dans tous les cas ci-dessus, la démarche utilisée est la même : découpage ou acquisition avec zone de recouvrement du document original, mise en correspondance des tuiles et fusion des informations qu'elles contiennent. Nous avons nous aussi adopté cette démarche en l'adaptant à nos besoins. En effet, les fusions décrites s'effectuent généralement sur des données *bitmap*, ce qui n'est pas notre cas. Seuls Janssen et Vossepoel effectuent une fusion à partir de données vectorielles. La méthode qu'ils mettent en œuvre, par recalage à partir de points de contrôle, est adaptée lors d'un calcul d'une transformation géométrique à appliquer aux différentes tuiles pour les recaler, ce qui n'est cependant pas notre cas. Nous avons donc développé une méthode de fusion qui s'applique au type de données (des segments) que nous manipulons et à la façon dont sont générées les différentes tuiles. Nous nous basons au départ sur le même principe que celui adopté par Vossepoel, son problème étant le plus semblable au nôtre. Notre *tuilage* comprend deux grandes phases [FIG. 4.1], développées par la suite :

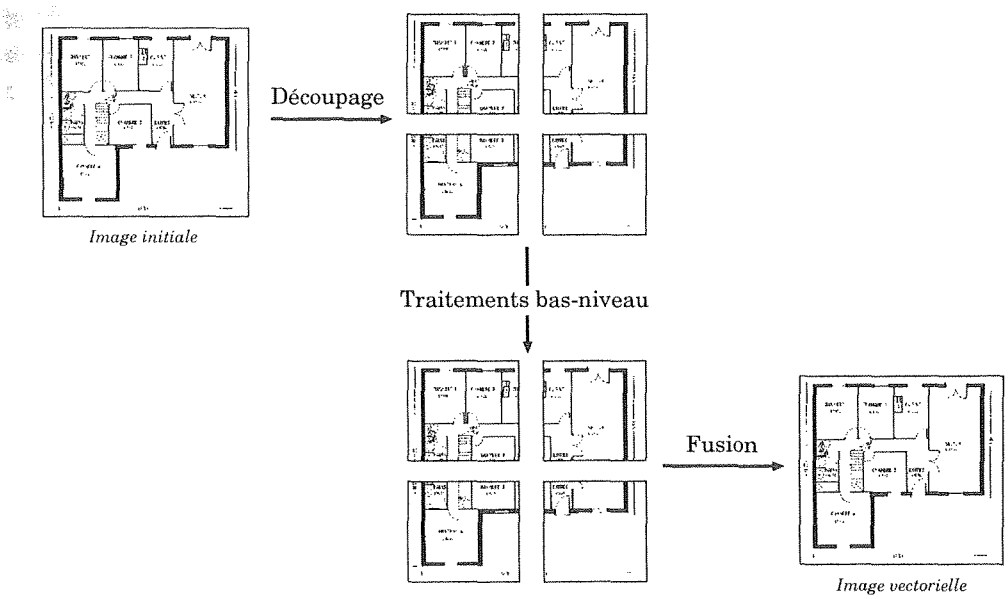


FIG. 4.1 – *Principe du tuilage. L'image originale est découpée en tuiles sur lesquelles nous appliquons les traitements bas-niveau. Les résultats sont fusionnés après la vectorisation, au moment où les tuiles sont manipulables sous forme d'ensembles de primitives graphiques haut-niveau.*

- Découpage des tuiles : cette étape permet de générer des tuiles de taille optimale par rapport aux contraintes imposées par les ressources disponibles sur une machine. Nous déterminons une zone

de recouvrement, dont la taille est fixée au mieux afin d'éviter tout problème lors de la phase suivante.

- Fusion des tuiles : après avoir appliqué les traitements bas-niveau sur chacune des tuiles, nous fusionnons les tuiles résultats (contenant des données vectorisées) afin de construire une image résultat globale. Cette fusion s'effectue à partir d'indices extraits sur chacune des tuiles. Nous mettons en correspondance les indices des différentes tuiles et nous reconstruisons ensuite les primitives sectionnées par le découpage.

4.4 Découpage des tuiles

Le découpage s'effectue sur l'image numérique originale, indifféremment avant ou après sa binarisation. Lors de cette étape, nous devons décomposer l'image en tuiles de taille plus raisonnable, mais également prendre garde à effectuer un découpage qui permette de fusionner ces tuiles à l'issue de notre analyse sans perdre d'information. À cette fin, nous définissons une zone de recouvrement, horizontale et verticale, entre les différentes tuiles pour pouvoir effectuer par la suite la mise en correspondance des primitives extraites de chacune d'elles. Nous avons finalement deux paramètres à déterminer pour cette étape : la taille des tuiles à générer ainsi que la taille de la zone de recouvrement nécessaire sur chaque tuile pour permettre une fusion ultérieure.

Pour déterminer la taille de la zone de recouvrement, nous calculons tout d'abord à partir de quelle distance du bord les informations contenues sur les tuiles sont consistantes. En effet, les traitements que nous appliquons engendrent des effets de bord et nous ne pouvons pas effectuer la fusion à partir de données non pertinentes. Vossepoel apporte une solution à ce problème [Vos 97]. Considérons le squelette des composantes connexes, décrit par des pixels p_i, comme l'endroit à partir duquel les composantes peuvent être reconstruites en utilisant des disques centrés sur les p_i, de rayon égal à l'épaisseur déterminée par la transformée de distance [§ 5.2.1]. Dans ce contexte, les pixels du squelette dont la position n'est pas altérée par la proximité du bord, c'est-à-dire les pixels dont le disque associé n'atteint pas le bord de la tuile, peuvent être considérés comme fiables. Ainsi, si l'épaisseur maximale de n'importe quelle composante connexe $E_{\max}$ est connue, nous sommes sûrs que les pixels du squelette se trouvant à $D = E_{\max}/2$ du bord de la tuile sont des pixels fiables du squelette. Bien entendu, D n'étant qu'une estimation — il faut vectoriser toute la tuile pour connaître l'épaisseur maximale exacte et nous ne sommes qu'au bas-niveau — il convient de définir une marge de sécurité. Vossepoel opte pour une taille $l > (1 + \varepsilon)E_{\max}$ pour cette première zone.

Finalement, cette première zone, dite de *recouvrement extérieure*, est d'abord définie selon chaque axe [FIG. 4.2]. Elle est peu fiable et n'est donc pas utilisée lors des mises en correspondance. C'est une deuxième zone, contiguë à la précédente, qui est utilisée. Celle-ci, dite de *recouvrement centrale*, doit également avoir une largeur suffisante pour que l'appariement soit fiable. Selon le même principe, sa largeur est fonction de $E_{\max}$ et est donc fixée à l. Enfin, pour être en mesure de superposer deux zones centrales appartenant à deux tuiles voisines selon un des axes, une troisième zone, dite de *recouvrement intérieure*, toujours de même largeur l, est nécessaire. La largeur totale L d'une zone de recouvrement est donc fixée à $3E_{\max} + \varepsilon$.

Les tuiles comportant des zones de recouvrement, certains pixels sont traités plusieurs fois : deux fois en général et jusqu'à quatre fois pour ceux qui se trouvent à l'intersection des zones horizontales et verticales. En posant :

- I_x, I_y : largeur et hauteur de l'image initiale,

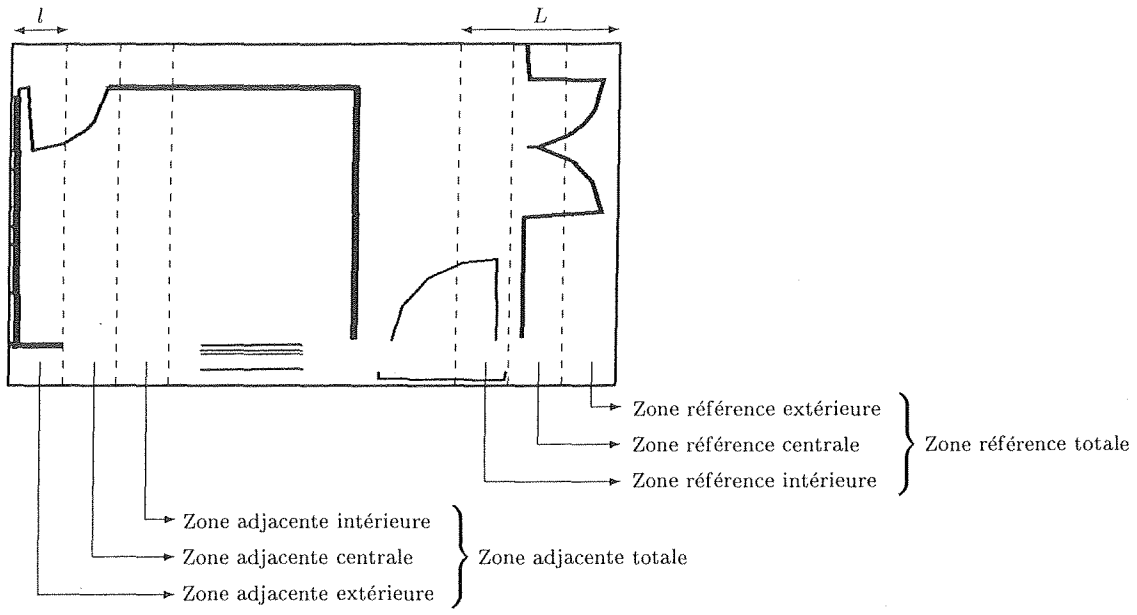


FIG. 4.2 – *Détail des différentes sous-zones d'une tuile. Les sous-zones présentées sont les sous-zones verticales. Le même découpage est effectué pour les sous-zones horizontales.*

- T_x, T_y : largeur et hauteur des tuiles,
- N_x, N_y : nombre de tuiles en largeur et en hauteur,

on calcule le pourcentage de pixels supplémentaires traités ρ comme

$$\rho = \frac{(I_x + L(N_x - 1))(I_y + L(N_y - 1)) - I_x I_y}{I_x I_y} \times 100$$

avec

$$N_x = \left\lceil \frac{I_x - T_x}{T_x - L} + 1 \right\rceil \quad (\text{si } I_x > T_x), \tag{4.1}$$

où la notation $\lceil\ \rceil$ correspond à la fonction d'arrondi supérieur. Nous pouvons poser une formule similaire à (4.1) pour N_y en considérant respectivement I_y et T_y à la place de I_x et T_x. Nous nous limitons dans la suite de ce paragraphe à une étude selon l'axe des abscisses, l'étude selon l'axe des ordonnées étant en tout point similaire.

Afin de minimiser ρ, l'unique variable qui reste paramétrable à ce stade est N_x, que nous ne pouvons moduler qu'à partir de T_x. Le meilleur rendement est ainsi obtenu en maximisant la taille des tuiles à générer, maximisation qui se fait selon les ressources mémoire disponibles de l'ordinateur effectuant les traitements. Mais encore une fois, nous ne pouvons fournir qu'une estimation de ces ressources, puisqu'il est difficile de connaître avec précision la quantité de mémoire qui sera effectivement disponible pour les données, la mémoire totale devant également contenir au moins le programme en lui-même et le système d'exploitation de la machine.

D'autre part, en imposant une largeur de tuile au moment du découpage, la largeur λ_x de la dernière tuile d'une ligne de tuiles est généralement différente de la largeur des autres tuiles. En posant $\mathcal{T}_x$ comme estimation de la largeur maximum d'une tuile, et en utilisant (4.1) pour calculer une estimation $\mathcal{N}_x$ à

partir de $\mathcal{T}_x$, cette largeur est égale à :

$$\lambda_x = I_x - (\mathcal{N}_x - 1)(\mathcal{T}_x - L).$$

Le découpage peut ne pas être optimal, surtout lorsque $\lambda_x \approx L$ (p. ex. si $\lambda_x = 102$ et $L = 100$, 100 pixels sont traités au moins deux fois au lieu d'une seule à cause de deux pixels). Pour éviter ce problème, nous répartissons les pixels de la dernière tuile qui se trouvent en dehors de sa première zone de recouvrement [FIG. 4.2] sur les tuiles précédentes. Ceci a pour effet de décrémenter N_x, donc de réduire ρ, et de rendre le découpage de la nouvelle dernière tuile optimal. La largeur T_x des tuiles devient alors :

$$T_x = \mathcal{T}_x + \left\lceil \frac{\lambda_x - L}{\mathcal{N}_x - 1} \right\rceil.$$

À partir de l'estimation de la mémoire maximale disponible $\mathcal{M}$ pour une tuile, nous calculons $\mathcal{T}_x = \mathcal{T}_y = \lceil \sqrt{\mathcal{M}} \rceil$, puis les valeurs corrigées T_x et T_y qui peuvent ne pas être égales entre elles. Ces valeurs corrigées sont utilisées si elles ne représentent pas une augmentation excessive de la taille ; le seuil est alors à déterminer en fonction des ressources disponibles. Dans le cas contraire, nous gardons la valeur initialement calculée pour la ou les tailles qui sont excessives. Les nombres finaux de tuiles en largeur (N_x) et en hauteur (N_y) sont ensuite obtenus grâce à (4.1).

4.5 Fusion des tuiles

Les tuiles sont traitées séparément, ce qui limite le volume des données à manipuler et rend aussi possible une parallélisation, sur un réseau de machines, voire sur une machine multiprocesseurs. Dans ce dernier cas, le gain se mesure en vitesse de calcul et non pas en espace mémoire. La fusion s'effectue à l'issue de la vectorisation et juste avant la détection d'arcs. Nous effectuons une fusion pour chacune des couches graphiques extraites de l'image initiale, c'est-à-dire généralement pour l'image de traits fins et pour l'image de traits forts. La fusion s'effectue alors à partir de l'ensemble de tuiles relatif à ces couches. Nous introduisons ici la terminologie relative à la fusion :

- Les objets contenus sur chacune des tuiles à l'issue de la vectorisation sont appelés *primitives graphiques*. Concrètement, ce sont des segments ; ils sont parfois désignés sous cette appellation.
- Lors de la mise en correspondance, les tuiles sont fusionnées deux à deux. Chaque tuile est considérée successivement par rapport à ses voisines, dont le nombre peut aller de 0 à 3. Dans cette mise en correspondance, la première tuile est appelée *tuile référence*. La seconde est appelée *tuile adjacente*.
- Les zones des tuiles à partir desquelles s'effectuent les mises en correspondance sont appelées *zones de recouvrement*. Il en existe selon chacun des axes, appelées zones de recouvrement horizontales et zones de recouvrement verticales. Chaque tuile — exception faite des tuiles se situant aux extrémités de l'image — contient des zones de recouvrement adjacentes, permettant de la mettre en correspondance avec la tuile précédente, et des zones de recouvrement références, permettant de la mettre en correspondance avec la suivante [FIG. 4.2]. Au total, ce sont ainsi 4 zones de recouvrement qui sont définies par tuile [FIG. 4.3].
- La mise en correspondance s'effectue à partir d'un sous-ensemble des primitives graphiques présentes sur chacune des tuiles, appelée *indices*.

Pour chaque couple de tuiles voisines (T_R,T_A), une référence et une adjacente, les candidats à l'appariement sont les primitives intersectant la zone centrale ou se trouvant entièrement dans cette zone.

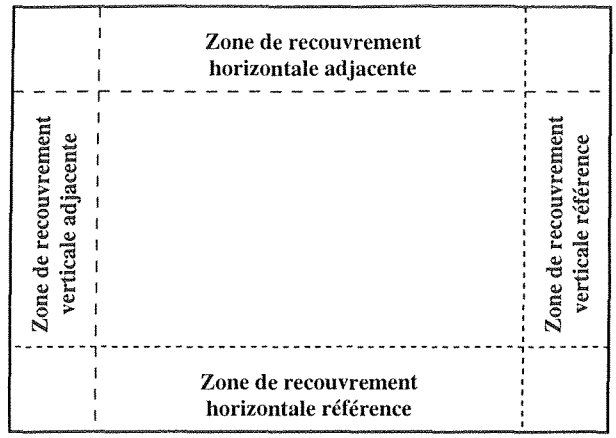


FIG. 4.3 – *Les quatre zones définies sur chaque tuile.*

Nous appelons *indices* ces primitives, dont nous ne considérons par la suite *que* la portion restreinte effectivement contenue dans la zone de recouvrement. Pour chaque appariement potentiel entre un indice R de la tuile de référence T_R et un indice A de la tuile adjacente T_A, un score est calculé en utilisant la distance de Hausdorff [Ruc 97, Hut 93], qui peut être définie comme :

$$H(R,A) = \max(h(R,A),h(A,R))$$

où :

$$h(R,A) = \max_{r \in R} \min_{a \in A} \|r - a\|,$$

$\|.\|$ représentant la distance euclidienne. La vectorisation étant effectuée, l'épaisseur des primitives est connue, et elles peuvent donc être assimilées à des parallélépipèdes. Les points r, respectivement a, parcourent les sommets du parallélépipède correspondant à l'indice R, respectivement A. Plus la distance est petite, meilleur est l'appariement. Si le score obtenu est inférieur à un seuil fixé expérimentalement (50 pour notre utilisation), A est considéré comme un bon candidat à l'appariement avec R.

Dans notre cas, chaque indice a un correspondant sur chaque tuile voisine à sa tuile d'appartenance. Pour stocker les appariements potentiels entre deux indices, nous utilisons des listes de type $L^X_{T_Y}$ qui contiennent les appariements potentiels d'un indice X avec les indices de la tuile T_Y. Les éléments de ce type de liste sont des couples comprenant :

- un indice I_Y de la tuile T_Y,
- le score d'appariement réalisé par le couple (X,I_Y).

Deux listes sont simultanément complétées lors de l'extraction d'un nouvel appariement (R,A) : la liste $L^R_{T_A}$ des indices adjacents de la tuile T_A pouvant correspondre à l'indice référence R et la liste $L^A_{T_R}$ des indices références de la tuile T_R pouvant correspondre à l'indice adjacent A. Nous effectuons l'extraction d'hypothèses d'appariements sur l'ensemble des couples de tuiles voisines. Toutes les listes utilisées sont ensuite triées selon un ordre croissant de score obtenu, ce qui permet d'obtenir les meilleurs correspondants en tête de liste. Jusqu'à six listes différentes peuvent être ainsi associées à chaque indice : trois pour l'indice en tant qu'indice référence et trois pour l'indice en tant qu'indice adjacent.

Nous effectuons ensuite une extraction des meilleurs correspondants. Nous considérons comme tuile de référence T_R toutes les tuiles une par une dans leur ordre d'apparition dans l'image, indifféremment

en largeur ou en hauteur d'abord. Nous recherchons alors le meilleur appariement possible pour chaque indice de T_R sur chacune des tuiles adjacentes. Cette recherche s'appuie sur une approche de type relaxation discrète [Moh 88], décrite par les algorithmes [ALG. 1] et [ALG. 2] (en pseudo-code objet). Ces algorithmes permettent d'extraire les paires de primitives (R,A) qui sont mutuellement meilleur

ALG. 1 – Tuile::ExtraitMeilleurCandidat()

```
1: encore = vrai
2: tant que encore == vrai faire
3:    encore = faux
4:    pour chaque segment s de la tuile faire
5:       si s.TrouveMeilleurAdjacent() alors
6:          encore = vrai
7:       fin si
8:    fin pour
9: fin tant que
```

ALG. 2 – Segment::TrouveMeilleurAdjacent()

```
RÉSULTAT: trouvé = faux
1: pour chaque tuile adjacente T_A faire
2:    si je ne suis pas apparié sur T_A et L^moi_{T_A}.Taille() ≠ 0 alors
3:       adjacent = L^moi_{T_A}.Tête()
4:       si adjacent.MeilleurModèle() == moi alors
5:          trouvé = vrai
6:          adjacent est marqué comme apparié sur la tuile à laquelle j'appartiens
7:          je suis marqué comme apparié sur T_A
8:          adjacent.MiseÀJourAdjacent(moi)
9:          MiseÀJourRéférence(adjacent)
10:      fin si
11:   fin si
12: fin pour
```

candidat l'une pour l'autre, c'est-à-dire pour lesquelles :

$$\forall P \in L^R_{T_A},\, H(R,A) \leq H(R,P) \quad \text{et} \quad \forall P \in L^A_{T_R},\, H(R,A) \leq H(P,A).$$

Lorsqu'un nouvel appariement est extrait, les listes stockant les appariements candidats sont mises à jour. L'algorithme [ALG. 3] décrit les opérations de mise à jour associées à la méthode `MiseÀJourAdjacent` de l'algorithme [ALG. 2]. Un algorithme similaire est associé à la fonction `MiseÀJourRéférence`, en permutant objets références et adjacents utilisés. Ces algorithmes permettent de supprimer toutes les occurrences des primitives extraites dans les listes.

ALG. 3 – Segment::MiseÀJourAdjacent(Segment $réf$)

1: atteindre la liste $L^{moi}_{T_R}$ telle que $réf \in L^{moi}_{T_R}$
2: **pour** chaque candidat r de $L^{moi}_{T_R}$ **faire**
3: atteindre la liste $L^{r}_{T_A}$ telle que $moi \in L^{r}_{T_A}$
4: suppression de moi dans $L^{r}_{T_A}$
5: **fin pour**
6: suppression de la liste $L^{moi}_{T_R}$

Une fois déterminés les meilleurs appariements pour toutes les paires possibles de tuiles voisines, les ensembles de primitives de toutes les tuiles sont fusionnés pour reconstituer l'image complète. L'algorithme [ALG. 4] considère tous les segments d'une tuile en leur cherchant des correspondants sur les tuiles voisines. Si le segment n'a pas de correspondant, il est ajouté tel quel dans la liste de segments résultat. Dans le cas contraire, l'algorithme [ALG. 5] recherche récursivement le correspondant

ALG. 4 – Tuile::Fusion()

RÉSULTAT: $listeSeg = \emptyset$
1: **pour** chaque segment s **non** utilisé de la tuile **faire**
2: s marqué comme utilisé
3: **si** $s \notin$ une des zones extérieures **alors**
4: **si** $s \in$ une zone centrale **alors**
5: $listeSeg$.Ajout(s.Fusion(s))
6: **sinon**
7: $listeSeg$.Ajout(s)
8: **fin si**
9: **fin si**
10: **fin pour**

d'un segment, ce qui permet de reconstituer les segments s'étendant sur plusieurs tuiles. La fonction `FusionFinale` permet de fusionner en un seul segment une liste de segments dont le premier et le dernier élément sont donnés en paramètre. La longueur du segment final est maximisée en respectant la position des extrémités, c'est-à-dire en conservant la connexité globale de la primitive. Ce dernier point est particulièrement important dans notre système, car la majorité des traitements de haut-niveau nécessitent cette information.

4.6 Résultats

Quelques résultats obtenus sont présentés [TAB. 4.1]. Nous utilisons des estimations $\mathcal{T}_x$ et $\mathcal{T}_y$ légèrement inférieures à la capacité réelle disponible, ce qui nous permet d'obtenir des tailles T_x et T_y plus optimales et également un peu plus grandes. Les deux dernières colonnes présentent le nombre et le

ALG. 5 – Segment::Fusion(Segment $segInit$)

RÉSULTAT: $segFinal$
1: **si** je possède un correspondant c **alors**
2: c est marqué comme utilisé
3: $segFinal$ = c.Fusion($segInit$)
4: **sinon**
5: **si** $moi = segInit$ **alors**
6: $segFinal = moi$
7: **sinon**
8: $segFinal$ = FusionFinale($segInit$, moi)
9: **fin si**
10: **fin si**

taux de segments qui ont un défaut de correspondant. Ceci traduit le fait qu'un segment qui devrait avoir un correspondant sur une tuile voisine n'en a pas. Ce genre de problème est généralement inhérent aux

I_x	I_y	T_x et T_y	L	T_x	T_y	ρ	Nb seg.	Défaut	% Défaut
6000	15600	2000	120	2000	2000	12,52	-	-	-
6000	15600	2000	120	2080	2055	9,60	2477	28	1,13
1812	3184	1000	200	1000	1000	45,08	-	-	-
1812	3184	1000	200	1006	1195	24,99	558	0	0,00
7200	16200	2000	120	2000	2000	11,22	-	-	-
7200	16200	2000	120	2480	2130	8,69	15189	60	0,39
2550	4200	1000	60	1000	1000	10,69	-	-	-
2550	4200	1000	60	1305	1095	6,74	13128	0	0,00

TAB. 4.1 – *Résultats du tuilage et de la fusion sur quelques exemples.*

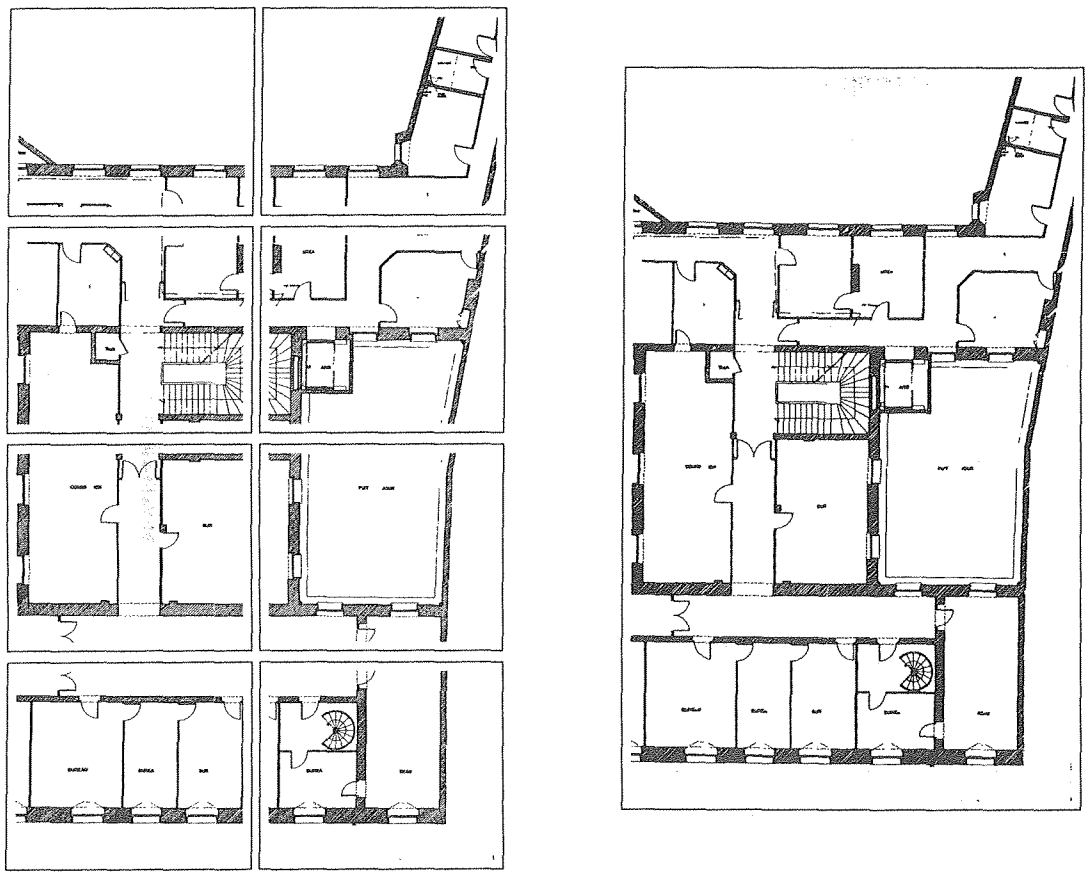
traitements qui sont appliqués sur les tuiles et à la qualité de l'image originale. La binarisation de Trier, et d'une façon générale les méthodes effectuant des traitements locaux ou calculant des seuils à partir de l'image globale avant d'effectuer leurs traitements, génèrent des résultats qui peuvent être différents pour une même zone, suivant la zone globale qui est traitée.

La méthode en elle-même donne de bons résultats dans ce contexte. Il y a plusieurs raisons à cela :

- Les tuiles proviennent d'une même image au départ. Les données présentes à un même endroit, par rapport à l'image originale, sont donc rigoureusement les mêmes sur différentes tuiles.
- Nous décidons du découpage à appliquer à l'image originale. Nous maîtrisons donc tous ses paramètres, et nous n'avons pas de problème pour recaler les différentes tuiles entres elles. Notre recalage ne nécessite par ailleurs pas de transformation à appliquer aux données contenues sur les tuiles (comme une translation, une rotation ou un changement d'échelle). Une telle transformation pourrait occasionner des imprécisions localement.

- Les données considérées dans la fusion n'étant pas proches du bord des tuiles, nous évitons la majeure partie des effets de bord qui accompagnent les traitements auxquels sont soumises les tuiles. Les résultats d'une même zone, toujours par rapport à l'image originale, sont donc généralement identiques sur différentes tuiles. Ce n'est cependant pas toujours le cas, notamment lorsqu'un critère est calculé en fonction de l'ensemble des données présentes sur une image. Dans ce cas, on peut obtenir des valeurs différentes suivant que l'on traite une image de façon globale ou par tuilage.

Un exemple de résultat de fusion est présenté [FIG. 4.4]. Ces résultats sont cependant difficilement appréciables visuellement en raison de leur grande dimension.



(a) Tuiles après binarisation.

(b) Fusion après vectorisation.

FIG. 4.4 – *Exemple de fusion de tuiles sur une image d'une taille initiale de* 2550×4200 *pixels. Les données quantitatives de cette fusion sont présentées en dernière ligne du tableau* [TAB. 4.1].

Troisième partie

De l'image brute à sa représentation en primitives graphiques

Chapitre 5

Vectorisation

5.1 Introduction

Jusqu'ici, nous n'avons manipulé au cours de notre analyse que des images numérisées. Ces images correspondent à des matrices de points attribués par une des deux couleurs que nous utilisons : le blanc pour désigner le fond de l'image et le noir pour désigner l'information contenue dans l'image. Les opérations de segmentations présentées [§ 3.3] ont permis de décomposer l'image binaire initiale en un certain nombre de couches, chacune contenant une partie de l'information initiale. Certaines couches correspondent directement à un type d'information recherchée, comme la couche de texte ou la couche graphique de traits forts qui décrit les murs porteurs d'un plan. D'autres, comme la couche graphique de traits fins, résultent en fait de la soustraction des couches désignées ci-dessus par rapport à l'image originale. On trouve sur cette couche les cloisons et les symboles architecturaux tels que les portes, les fenêtres, les escaliers...

Quelle que soit la couche considérée, les symboles architecturaux ne sont pas encore clairement identifiés. On sait par exemple qu'une image contient la représentation des murs porteurs, mais il est encore impossible de connaître leur nombre, leur position et d'autres attributs tels que leur épaisseur. En effet, les images de points ne véhiculent qu'une information sémantique très faible qui ne permet pas de distinguer de tels symboles, même s'ils y sont représentés. Afin d'atteindre une abstraction supplémentaire et de pouvoir travailler avec des entités plus symboliques que les points, nous effectuons une conversion des images de points en images de vecteurs. Cette opération est appelée *vectorisation.*

La littérature est assez abondante sur les méthodes de vectorisation, cette étape se trouvant au cœur de tout système d'analyse de documents. Ceci nous a d'ailleurs amené à implanter plusieurs méthodes dans notre plateforme, et continue actuellement à entretenir des recherches au sein de notre équipe. Notre objectif est de nous doter de méthodes assez robustes, fonctionnant sur le principe des « boîtes noires », afin de pouvoir les réutiliser par la suite. Nous pensons qu'un des facteurs permettant de parvenir à cette robustesse est de limiter le nombre de paramètres et de seuils de cette étape [Tom 98a, Tom 99], peut-

être plus encore que pour les autres traitements présentés dans ce document. En effet, de nombreuses vectorisations sont trop dépendantes d'une multitude de paramètres délicats à régler.

La plupart des méthodes de vectorisation sont composées de plusieurs étapes [Tom 99] :

- l'extraction des lignes (droites ou courbes) présentes dans l'image originale, dont l'approche la plus courante est d'extraire le squelette de la partie graphique de l'image [§ 5.2] ;
- l'approximation des lignes extraites en vecteurs, généralement des segments, quoique certaines méthodes proposent directement des approximations fournissant des segments et des arcs [§ 5.3] ;
- l'application de post-traitements permettant de repositionner les points de jonctions des primitives vectorielles extraites, de les fusionner ou d'en supprimer une partie selon certains critères ;
- l'extraction des arcs à partir des segments si cette étape n'a pas été réalisée auparavant. Dans ce mémoire, cette étape est décrite chapitre 7.

Ces étapes ne sont pas toujours rigoureusement suivies par toutes les méthodes de vectorisation. Nous avons toutefois privilégié l'intégration dans la bibliothèque ISADORA (cf. chapitre 14) des méthodes basées sur cette décomposition, pour des raisons de robustesse, de modularité et de réutilisabilité.

5.2 Extraction des lignes

La première étape de la vectorisation consiste à extraire un ensemble de lignes de l'image. Intuitivement, ces lignes correspondent à l'axe médian des traits[17] contenues dans l'image et reflètent ainsi sa structure. Afin de déterminer ces lignes, trois grandes familles de méthodes ont été proposées :

- La première et principale famille se propose de calculer l'axe médian lui-même, c'est-à-dire le squelette de l'image [§ 5.2.1], par l'utilisation d'une transformation mathématique. Ces méthodes sont relativement simples, fonctionnent sur n'importe quel type de documents, mais fournissent souvent des résultats dégradés (barbules, mauvais positionnement des points de jonction).
- La deuxième famille est basée sur l'appariement des contours extérieurs [§ 5.2.2]. Si les résultats obtenus sont de meilleurs qualités, ces méthodes dépendent d'heuristiques et de seuils qui ne sont pas toujours évidents à déterminer, et ne sont ainsi pas applicables sur tout type de documents.
- Enfin, la troisième famille [§ 5.2.3] regroupe des méthodes qui ne considèrent pas tous les points d'une image, mais utilisent des sous-échantillonnages pour extraire grossièrement les lignes de l'image.

5.2.1 La squelettisation

Le but de la *squelettisation* est de réduire les traits présents dans une image binaire à une ligne d'une épaisseur d'un pixel, appelé *squelette*. Cela permet d'extraire l'essentiel de l'information décrite par l'image : sa structure. À partir de cette nouvelle description de l'image, plus compacte et ainsi plus expressive sémantiquement, le calcul des vecteurs décrivant l'image se trouve facilité. Toutefois, si le squelette obtenu permet de traiter l'image plus facilement, il est impératif qu'il ne dénature pas l'information présente dans l'image initiale. Le calcul du squelette doit donc garantir que les propriétés topologiques et morphologiques de l'image initiale, telles que la connexité, les jonctions et les dimensions, seront préservées à l'issu du traitement.

17. Nous appelons *traits* les parties graphiques représentées par une image numérisée. Ces traits ne doivent pas être confondues avec les *primitives vectorielles* (segments, arcs) qui résultent de la vectorisation.

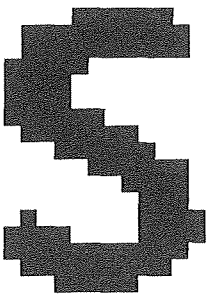
Les méthodes itératives

Les méthodes les plus classiques de squelettisation considèrent son calcul par amincissements successifs [Lam 92, Lam 93]. À chaque itération, les traits de l'image sont « rabotés » : une épaisseur d'un pixel est supprimée au niveau de leurs contours, à la condition toutefois que les propriétés des traits ne soient pas altérées. Pour cela, chaque point de l'image est considéré à chaque itération, ainsi que son voisinage immédiat. Le point est alors supprimé s'il correspond à l'une des configuration de la figure [FIG. 5.1]. Le traitement est itéré jusqu'à l'obtention d'une image stable, c'est-à-dire dont on

0	0	0	1	x	0	1	1	1	0	x	1
x	1	x	1	1	0	x	1	x	0	1	1
1	1	1	1	x	0	0	0	0	0	x	1
x	0	0	x	1	x	x	1	x	0	0	x
1	1	0	1	1	0	0	1	1	0	1	1
x	1	x	x	0	0	0	0	x	x	1	x

FIG. 5.1 – *Éléments structurants conditionnant la suppression d'un point de l'image lors de la squelettisation par amincissements successifs. Dans ces éléments, 0 représente un pixel blanc, 1 un pixel noir et x un pixel indifféremment blanc ou noir.*

ne peut plus retirer de pixel sans remettre en cause les propriétés topologiques et morphologiques de l'image. L'image finale décrit alors le squelette du contenu de l'image initiale. Afin de garder une trace de l'épaisseur initiale des traits, chaque point effacé de l'image est étiqueté par le numéro de l'itération qui a permis de le supprimer [FIG. 5.2]. À la fin de la squelettisation, les points du squelette sont étiquetés par la somme maximale des étiquettes de deux de leurs voisins symétriques (quatre couples existent potentiellement : le vertical, l'horizontal et les deux diagonaux).



```
      . 1 1 1 1 .
  . 1 1 . . . . . . 1
  . . . 2 1 1 1 1 1 .
2 . 3 2 1
1 . 3 2
1 . 3 2
. . . . 2 1
  1 2 3 . . 2 1
      1 2 3 . 2 1
          1 2 . . . .
              1 2 . 1
                  1 . 1
  1               1 . 2 1
1 2 . 1           1 . . .
. . . 2 1 1 1 1 2 . 1
  1 2 . . . . . . 1
      2 1 1 1 2
```

FIG. 5.2 – *Image d'un caractère et étiquetage correspondant après squelettisation par amincissements successifs.*

Ces méthodes sont relativement simples à mettre en œuvre, mais présentent certains inconvénients :

- Elles sont assez coûteuses en temps d'exécution. D'une part, le nombre d'itérations, de *passes*, à réaliser dépend directement de la valeur de l'épaisseur maximale des traits du document original. Plus cette valeur est élevée et plus la convergence nécessite de passes pour être atteinte (ce qui peut encore être accentué avec la résolution de numérisation). D'autre part, ces méthodes sont

implantées par application de masques sur l'image : tous les points de l'image sont donc considérés à chaque passe.
Des implantations parallèles ont cependant été proposées, ce qui permet de réduire la complexité du traitement [Che 93b].

- Elles sont assez sensibles au bruit, notamment aux irrégularités présentes à la surface des contours. Lors des amincissements successifs, ces irrégularités se propagent et provoquent des barbules et des erreurs de positionnement des jonctions. Cela implique donc l'utilisation de post-traitements permettant de corriger ces artefacts.

Ces inconvénients nous ont amené à tester d'autres méthodes de squelettisation afin d'améliorer les performances et la robustesse de notre vectorisation.

La transformée de distance 3–4

Pour obtenir des résultats de meilleure qualité, nous sommes tournés vers une méthode à base de *transformée de distance*, plus satisfaisante au niveau de la qualité [Bor 86]. L'objectif recherché par cette méthode est de mieux traiter les points de jonction et de limiter le nombre de barbules et autres artefacts. Intuitivement, son principe est de considérer que le squelette de l'image est déterminé par le centre des cercles maximaux inscrits dans les traits de l'image. L'implantation est réalisée de la manière suivante :

1. Calcul de la transformée de distance. Chaque point noir d'une image binaire est étiqueté par la distance entre ce point et le point blanc le plus proche. Pour approcher au mieux la métrique euclidienne tout en gardant des temps de calcul faibles, des métriques de type *chanfrein* sont habituellement utilisées ; dans notre cas, nous avons adopté la métrique dite 3–4 [dB 94], dont l'algorithme est donné ci-dessous, avec la première passe sur l'image de gauche à droite et de haut en bas, et la deuxième passe de droite à gauche et de haut en bas :
   - première passe : $Y = \min(V_1 + 3, V_2 + 4, V_3 + 3, V_4 + 4)$,
   - deuxième passe : $Z = \min(Y, V_5 + 3, V_6 + 4, V_7 + 3, V_8 + 4)$.

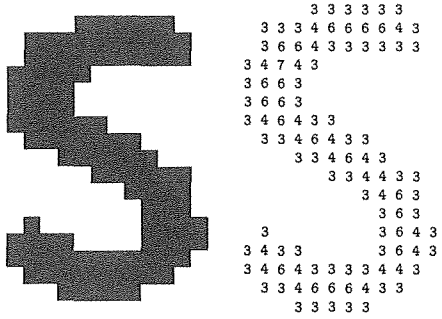


FIG. 5.3 – *Image d'un caractère et sa TD 3–4.*

V_2	V_3	V_4
V_1	X	V_5
V_8	V_7	V_6

FIG. 5.4 – *Voisinage utilisé pour le calcul de la TD 3–4.*

2. Détection des maxima locaux. À partir de la transformée de distance, nous détectons les pixels appartenant au squelette. Nous nous basons pour cela sur un algorithme proposé par Arcelli et Sanniti di Baja [Arc 85, Arc 89]. Nous recherchons d'abord tous les pixels qui sont centres d'un disque maximal. Pour ce faire, on remplace d'abord, dans la transformée de distance, les étiquettes 3 et 6 par les étiquettes 1 et 5. Un pixel est alors le centre d'un disque maximal si $V_i < X + 3$ pour tout i impair, et $V_i < X + 4$ pour tout i pair.

Tout centre de disque maximal X détecté à ce niveau et ayant trois voisins consécutifs V_i,V_{i+1},V_{i+2} (addition modulo 8), avec i impair, tous supérieurs à X, n'est pas marqué. Les pixels qui ne sont pas des centres de disques maximaux et qui ont plus d'une composante connexe de voisins dont les étiquettes sont supérieures à X, sont marqués comme appartenant au squelette. Les pixels d'étiquette 3 ayant trois voisins consécutifs V_i,V_{i+1},V_{i+2} (addition modulo 8), avec i impair, tous étiquetés 3, sont également marqués.

3. Suivi du squelette. Il s'agit ensuite de prolonger le squelette en reliant les points ainsi marqués. Pour ce faire, on trace le squelette dans le sens du gradient maximal, à partir des pixels déjà marqués. Ce gradient est calculé comme suit :

$$grad(V_k) = \frac{1}{w_m}(V_k - X)$$

avec :

$$w_m = \begin{cases} 3 \text{ si } k \text{ impair} \\ 4 \text{ sinon.} \end{cases}$$

Pour chaque pixel X marqué, on s'oriente dans la direction de son voisin V_k non marqué et dont le gradient est maximal. On marque ce dernier et on réitère l'opération en se positionnant sur V_k.

4. Réduction à l'épaisseur unité. Le résultat ainsi obtenu est réduit à l'épaisseur 1 sans altérer la connectivité du squelette, ni raccourcir ses branches.
Pour cela, on efface la marque des pixels ayant au moins un voisin V_i, avec i impair, non marqué, et au moins un triplet de voisins V_i,V_{i+2},V_{i+5} (i impair, addition modulo 8) tel que V_i et V_{i+2} sont marqués tandis que V_{i+5} n'est pas marqué.
5. Ébarbulage. La technique d'ébarbulage proposée par Sanitti di Baja est intéressante dans la mesure où elle ne se réduit pas à couper les barbules de longueur inférieure à un seuil, mais permet de fixer un seuil d'importance de la barbule, qui est fonction de la valeur de la transformée de distance aux points supprimés, ce qui limite l'erreur de reconstruction introduite par cette opération [dB 94].
L'algorithme consiste à partir de toutes les extrémités libres, et à marquer comme points potentiellement effaçables ceux qui sont de degré 2 et dont la contribution à la représentation de la forme d'origine est inférieure au seuil fixé. Ce seuil est le seul paramètre de la méthode de squelettisation ; sa valeur par défaut est de zéro, ce qui supprime dans ce cas la phase d'ébarbulage.

Afin de gérer l'épaisseur initiale des traits, nous nous basons sur le squelette, étiqueté par la valeur de la transformée de distance [FIG. 5.5]. Cette valeur est multipliée par 2/3 afin d'obtenir l'épaisseur finale, la transformée de distance 3–4 ne correspondant pas à la distance euclidienne.

La figure [FIG. 5.6] présente les résultats obtenus par cette méthode de squelettisation, en les comparant à ceux de la méthode « classique » par amincissements successifs. Les résultats obtenus sont de meilleure qualité : la transformée de distance 3–4 est moins sensible au bruit et est plus performante en temps d'exécution, ne dépendant pas de l'épaisseur maximum des traits présents. Des imprécisions sont toutefois générées, notamment au niveau des jonctions. Des améliorations possibles sont proposées à la fin de ce chapitre [§ 5.4].

La qualité des résultats est satisfaisante sur des images ne comportant pas de traits trop épais — typiquement les images de traits fins issues de la segmentation traits forts / traits fins. En revanche, la qualité des résultats se dégrade nettement sur les images de traits forts [FIG. 5.7]. Si la transformée de distance 3–4 fournit de meilleurs résultats que les méthodes de squelettisation par itérations, elle montre cependant ses limites sur ce type de données. La recherche des centres maximaux des cercles inscrits dans les traits de l'image produit des artefacts aux extrémités des lignes, dont l'importance est proportionnelle à l'épaisseur des traits.

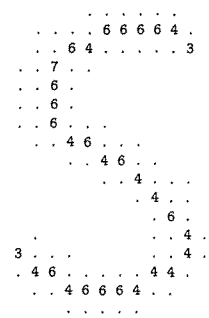
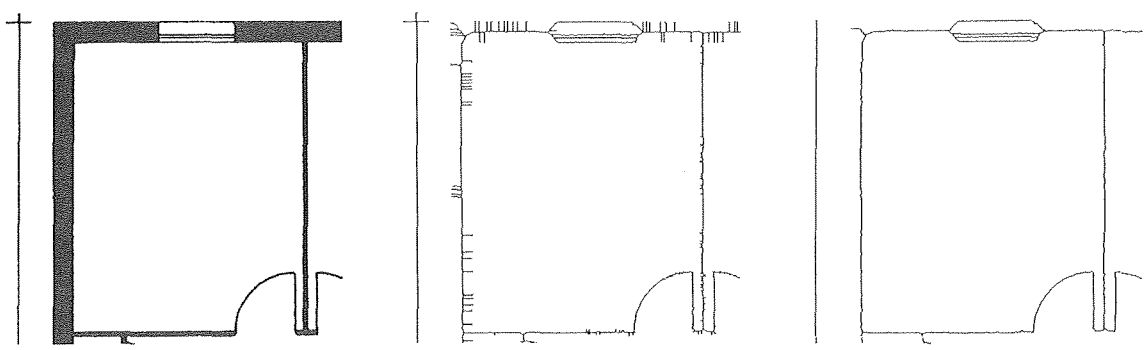
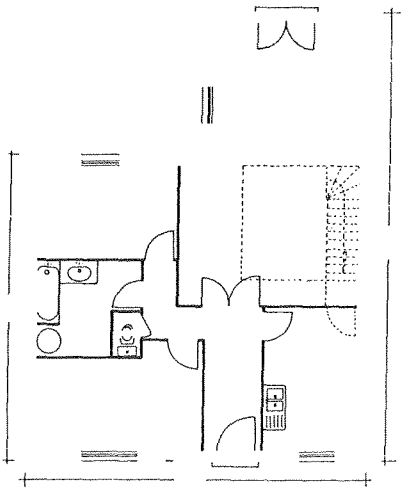


FIG. 5.5 – *Valeur de la transformée de distance 3–4 au points du squelette extrait du caractère de l'image [*FIG. *5.3].*

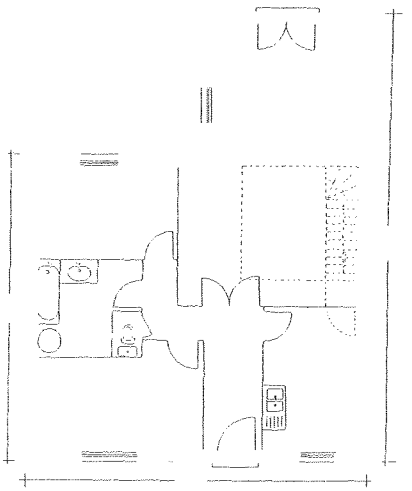


(a) Extrait de l'image originale. (b) Squelette classique. (c) Squelette TD 3–4.

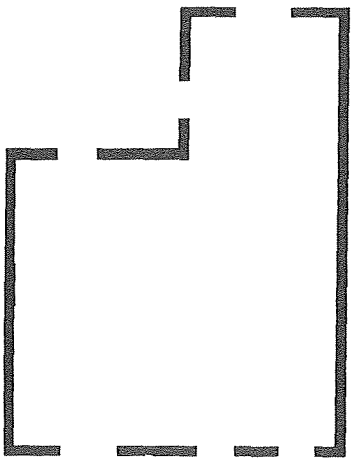
FIG. 5.6 – *Comparaison entre les deux méthodes de squelettisation.*



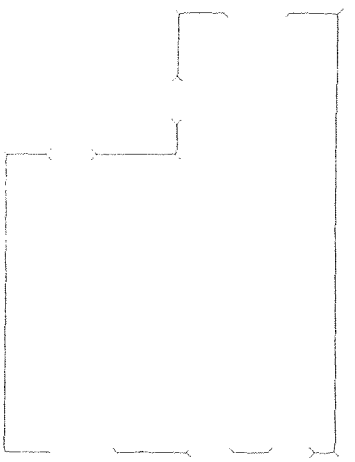
(a) Image originale de traits fins.



(b) Squelette TD 3–4.



(c) Image originale de traits forts.



(d) Squelette TD 3–4.

FIG. 5.7 – *Résultats obtenus sur des images de traits fins et de traits forts. La transformée de distance 3–4 montre ses limites lorsque l'épaisseur des traits devient importante : des barbules apparaissent alors aux extrémités des lignes.*

Chaînage

Une fois le squelette calculé, nous passons d'une représentation sous forme d'une image de points à un ensemble de *chaînes* de points. Chacune de ces chaînes décrit une partie du squelette, et est constituée de la liste des points connexes d'une même « branche » du squelette. L'algorithme que nous avons implanté pour réaliser ce *chaînage* est le suivant [Tab 94, Tom 99] :

1. Marquage de tous les points du squelette qui sont des extrémités libres ou qui sont à une jonction entre plusieurs branches ; ce marquage se fait très simplement par examen du voisinage 3×3 de chaque pixel.
2. À partir de chaque point ainsi marqué, effacement du point s'il est isolé (degré égal à 0), sinon :
   - mémorisation de la liste de tous les voisins du point qui sont le début potentiel d'une branche : on traite d'abord les voisins 4-connexes, puis on ajoute les voisins 8-connexes qui ne sont pas connexes avec un voisin 4-connexe déjà traité ;
   - à partir de chacun des points de cette liste de voisins, démarrage du suivi d'une branche ; on cherche prioritairement un voisin 4-connexe non marqué, à défaut un voisin 4-connexe marqué, à défaut un voisin 8-connexe non marqué, à défaut un voisin 8-connexe marqué, à défaut un autre point de la liste initiale des débuts de branche (cas d'une chaîne en boucle) ;
   - tant qu'on peut poursuivre le parcours vers un voisin non marqué, itération du processus, en effaçant au fur et à mesure les pixels suivis non marqués ;
   - arrêt du suivi sur un autre point marqué (cas d'une branche ouverte) ou sur un autre point de la liste initiale des débuts de branche (cas d'une boucle).
3. Quand toutes les branches partant du point ont été suivies, effacement du point.
4. Une fois que toutes les branches partant de tous les points marqués ont été suivies, il peut encore rester des circuits (boucles) sans aucun point marqué. Pour les chaîner, on parcourt l'image en cherchant un point encore non traité, et on lance le processus de suivi de la boucle correspondante chaque fois qu'on en trouve un.

Cet algorithme est présenté plus en détail [ALG. 6]. À l'issue de cette étape, une description de l'information contenue dans l'image analysée est donc disponible sous forme d'un ensemble de chaînes de points, correspondant aux différentes branches du squelette. C'est la première fois depuis le début de l'analyse que l'information n'est plus manipulée sous forme d'une matrice de points. Outre la signification sémantique plus forte de ce type de représentation, un des avantages induits est le gain en espace mémoire. En effet, seuls les points correspondants à l'information sont maintenant stockés.

5.2.2 Appariement de contours

Les méthodes à base d'appariement de contours permettent également d'extraire les lignes d'une image. Le principe de ces méthodes est de calculer les contours des traits de l'image, d'apparier ensuite les contours qui sont opposés, ce qui permet de déterminer l'axe médian des traits [Han 94]. La faiblesse de ce type d'approche réside dans l'appariement des contours opposés. En effet, si l'appariement est relativement aisé sur des formes de traits simples, à base de parallélépipèdes par exemple, il devient beaucoup plus délicat sur des formes plus complexes, telles que des courbes, des formes comprenant des jonctions multiples, ...

Ce type de méthodes a été expérimenté dans notre équipe il y a quelques années lors de la conception du système REDRAW [Ant 92]. Afin de pouvoir vectoriser proprement les traits épais, nous avons implanté une autre méthode plus récemment [AS 98b]. Basée sur des connaissances *a priori*, cette méthode

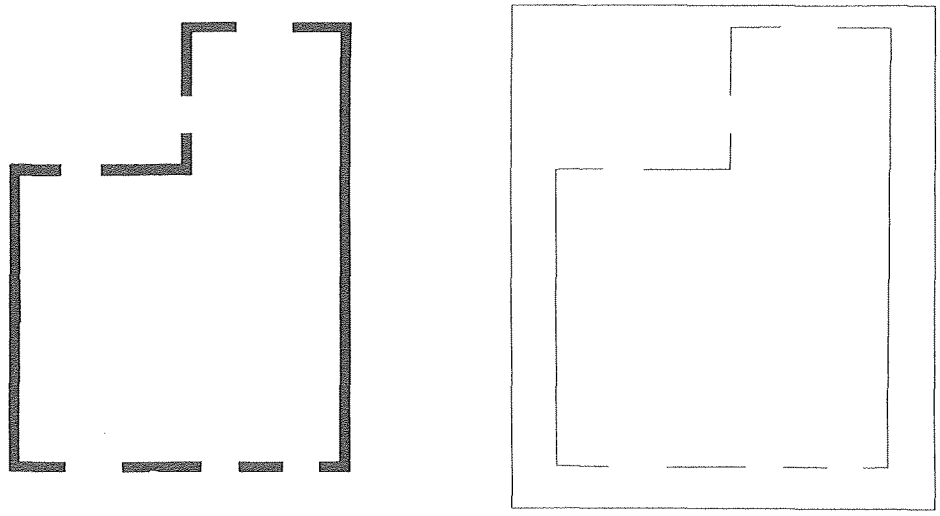
ALG. 6 – Chaînage des pixels d'un squelette

```
1: Marquer tous les pixels de degré ≠ 2, ou de degré = 2 quand 4 pixels forment un carré
2: Supprimer les pixels isolés
3: tant que il reste des points marqués faire
4:    Choisir un point marqué p
5:    Créer une liste l_p de tous les voisins non nuls de p
6:    tant que l_p ≠ ∅ faire
7:       Choisir un point q dans l_p
8:       Mettre p temporairement à 0 pour éviter des boucles prématurées
9:       Créer une nouvelle chaîne c initialisée à [p,q]
10:      si q est marqué alors
11:         continueChaine ← faux
12:      sinon
13:         continueChaine ← vrai
14:         Mettre q à 0
15:      fin si
16:      tant que continueChaine = vrai faire
17:         q ← un4VoisinNonMarqué(q)
18:         si aucun n'est trouvé alors
19:            q ← un4VoisinMarqué(q)
20:            si aucun n'est trouvé alors
21:               q ← un8VoisinNonMarqué(q)
22:               si aucun n'est trouvé alors
23:                  q ← un8VoisinMarqué(q)
24:                  si aucun n'est trouvé alors
25:                     q ← unVoisinQuelconque(q)
26:                  fin si
27:                  continueChaine ← faux
28:               fin si
29:            sinon
30:               continueChaine ← faux
31:            fin si
32:         fin si
33:         Ajouter q à c
34:         si continueChaine = vrai alors
35:            Mettre q à 0
36:         sinon si nous sommes sur une boucle alors
37:            Ajouter p à c à nouveau pour boucler la boucle
38:         fin si
39:      fin tant que
40:      Ajouter c à la liste de chaînes
41:      Restaurer la marque de p
42:   fin tant que
43:   Mettre p à 0 (il est complètement traité)
44: fin tant que
45: tant que il reste des pixels non traités (les circuits) faire
46:    Recommencer tout le processus après avoir marqué un pixel non nul
47: fin tant que
```

ne se veut pas universelle, mais se propose d'apporter une solution ponctuelle pour le type d'images que nous manipulons, les traits épais des plans architecturaux dont nous disposons étant relativement simples à modéliser. En particulier, la méthode implantée ne s'intéresse qu'aux jonctions en « T » ou en « L ». La méthode est basée sur l'étude du contour des composantes connexes. L'appariement de contours opposés permet ensuite de calculer l'axe médian du trait. Cette méthode se décompose en plusieurs étapes :

1. recherche du contour des traits : un pixel noir appartient au contour si au moins un des ses pixels voisins 4–connexes est blanc,
2. chaînage des contours obtenus, et approximation polygonale par l'une des méthodes proposées par la suite [§ 5.3],
3. post-traitements permettant de réduire le bruit sur les résultats obtenus :
   - fusion en un seul segment des segments connexes dont l'angle est proche de 180°,
   - connexion des paires de segments non connexes présentant des extrémités relativement proches l'une de l'autre ; les extrémités de ces segments sont alors confondues à l'emplacement du centre du segment défini par la position originale des extrémités,
4. recherche des axes médians à partir des segments obtenus ; quelques connaissances *a priori* injectées pour modéliser les segments simples permettent d'obtenir ces axes, et finalement les segments correspondants aux lignes de l'image.

Cette méthode donne de bons résultats sur les images de traits forts que nous manipulons [FIG. 5.8]. Elle reste cependant limitée, ne permettant pas de traiter des jonctions plus complexes que celles modélisées. Il est possible de l'améliorer, mais nos recherches plus récentes se tournent vers d'autres pistes pour résoudre le problème délicat de la vectorisation [§ 5.4].



(a) Image originale de traits forts.

(b) Image vectorisée.

FIG. 5.8 – *Résultat de la vectorisation par appariement de contours d'une image de traits forts.*

5.2.3 Approches non-systématiques

L'idée de base de cette troisième famille de méthodes est de ne pas considérer tous les points d'une image, mais de se baser uniquement sur certains d'entre eux pour obtenir une vue d'ensemble du dessin à traiter. On trouve dans cette catégorie des méthodes plus originales, comme OZZ[18] et SPV[19] de Dori et Wenyin [Dor 93, Wen 96, Wen 98b]. Le principe de ces méthodes s'assimile à un lancer de rayon : l'image originale est parcourue par un rayon, selon une direction parallèle aux axes de l'image, jusqu'à la rencontre d'un point noir, point d'entrée dans un trait. Le rayon est ensuite prolongé jusqu'à la rencontre d'un point blanc. La trace du rayon à l'intérieur du trait est alors utilisée pour poursuivre la méthode : à partir du centre de cette trace, un second rayon est lancé dans une direction orthogonale au premier. Le processus est itéré jusqu'à ce que le bout de la composante connexe soit atteint (moyennant un seuil). La suite des centres des traces permet alors de déterminer l'axe médian du trait parcouru. Cette méthode fournit des résultats intéressants, mais ne préserve pas toujours la connexité globale du document.

Notre équipe s'est également intéressé à une approche de ce type lors de la conception du système CELESSTIN [Vax 95], en se basant sur des travaux de Lin *et al.* [Lin 85]. Le principe de cette méthode est de décomposer l'image initiale en *mailles* de taille donnée. Cette taille doit être choisie afin qu'une maille ne contienne au plus qu'un trait de l'image initiale. L'extraction des lignes est ensuite effectuée en confrontant le bord des mailles à l'une des 48 configurations proposées par Lin *et al.* [FIG. 5.9]. À

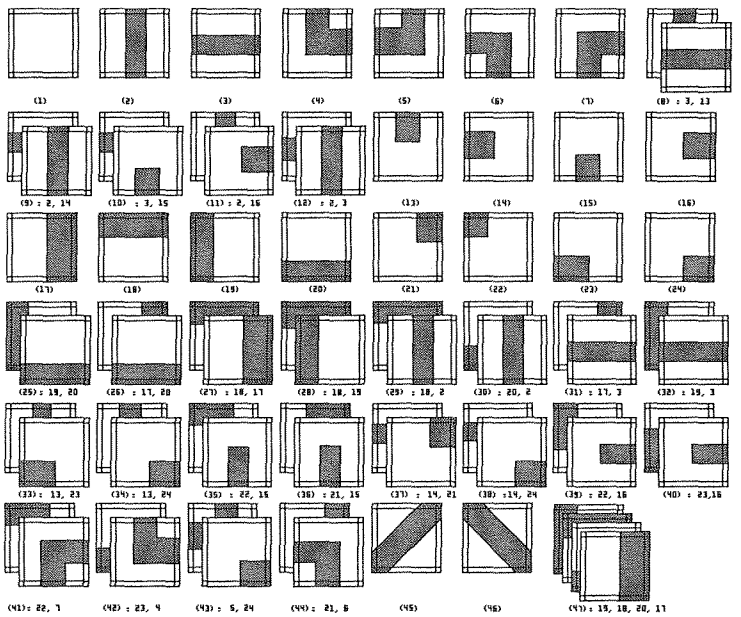


FIG. 5.9 – *Les configurations permettant d'analyser le bord des mailles [Lin 85].*

partir des identifications effectuées, des règles structurelles permettent ensuite de retrouver les lignes de l'image. Les résultats sont assez satisfaisants, mais une fois de plus la connexité globale de l'image n'est pas toujours respectée et le choix de la taille des mailles est parfois difficile à déterminer.

D'autres méthodes non-systématiques ont été proposées, comme la décomposition de l'image initiale en *plages*, des suites de points noirs connexes dans une ligne[20] ou une colonne [Zen 96]. La recherche

18. *Orthogonal Zig-Zag.*
19. *Sparse Pixel Vectorization.*
20. Il s'agit ici d'une ligne de l'image perçue comme une matrice de points.

des lignes s'effectue ensuite en étudiant les relations existantes entre ces plages, menant à la construction de graphes d'adjacence (un pour les champs horizontaux et un pour les champs verticaux). L'axe médian est alors calculé à partir de ces graphes et d'heuristiques. La méthode est également intéressante, mais les résultats sont conditionnés par la configuration des heuristiques.

5.2.4 Synthèse

Aucune des approches proposées ci-dessus ne permet de cumuler les différentes qualités attendues de l'extraction de lignes : une localisation précise de la ligne, des points de jonctions et un réglage simple de la méthode qui permette de l'utiliser de manière générique. En effet :

- les méthodes à base de squelettisation permettent d'obtenir un bon taux de localisation, mais sont relativement sensibles au bruit. De plus, il y a souvent des distorsions au niveau des intersections et des jonctions ;
- les méthodes à base d'appariement de contours présentent souvent des meilleurs taux de localisation de l'axe médian et sont aussi plus performantes sur la localisation des points de jonctions. Mais elles sont aussi plus délicates à régler, reposant sur plus d'heuristiques ou de seuils que les méthodes de squelettisation ;
- les approches non-systématiques fournissent des résultats d'assez bonne qualité, mais ne permettent pas toujours d'exploiter les petits détails présents. Elles reposent également sur des heuristiques et seuils parfois difficiles à déterminer.

Ceci a conduit à considérer des méthodes combinant plusieurs approches. C'est le cas notamment de Hori et Okazaki [Hor 92] qui proposent l'utilisation d'une méthode à base de squelettisation pour localiser les lignes, et d'un appariement de contours pour réduire le bruit et avoir un meilleur positionnement des points de jonctions. Pour nos besoins, nous avons choisi de nous baser sur la transformée de distance 3–4 et sur la méthode d'appariement de contours présentée [§ 5.2.2] suivant le type de données à traiter. Nous sommes cependant conscients que ces méthodes ne sont pas parfaites, et nous continuons à développer nos modules de recherche de lignes, et plus généralement de vectorisation.

Enfin, il est important d'être attentif aux méthodes proposées dans des domaines voisins du nôtre. Même si ces méthodes restent trop expérimentales ou trop coûteuses lorsqu'elles sont appliquées sur nos données, elles peuvent devenir matures à terme et proposer des alternatives intéressantes aux techniques actuellement utilisées. En particulier, il existe d'autres approches que celles basées sur une transformation mathématique. C'est le cas de la méthode de Zhu [Zhu 99], qui pose le problème de la recherche des lignes comme un problème d'inférence statistique. Il propose un algorithme stochastique qui, à partir d'une forme, génère plusieurs hypothèses d'axe médian, chacune associée à une probabilité. L'hypothèse retenue est finalement sélectionnée grâce à l'utilisation de modèles statistiques décrivant les alignements, les différentes jonctions (en « T », en « X », en « Y »). Les résultats présentés sont intéressants, aussi bien sur des traits tels que ceux utilisés dans les documents techniques, que sur des formes plus complexes comme des dessins de silhouettes d'objets ou d'organismes vivants. Ce type de méthodes est cependant très coûteux en temps de traitement et ne peut pas encore être envisagé pour traiter une image entière. Au mieux, il peut compléter l'étude locale d'une image.

5.3 Approximation polygonale

L'objectif de l'*approximation polygonale* est de représenter les lignes, disponibles sous forme de listes de points, par une suite de segments sans trop dénaturer la forme originale des lignes. Les méthodes d'approximation sont généralement paramétrées par un seuil qui fixe la précision de l'approximation. Plus cette précision est grande, plus le nombre de segments générés est grand. La valeur seuil représente donc un compromis entre la précision et la quantité de segments nécessaires à l'approximation.

Il y a un double avantage à effectuer ce traitement. Tout d'abord, au niveau de l'espace mémoire, puisqu'une liste de points est représentée, suite à l'approximation polygonale, par une liste nettement moins importante de segments. Ensuite, au niveau de l'abstraction symbolique : les segments représentent une information plus pertinente que les chaînes de points. Un seul segment permet en effet de représenter un groupe de points, et est de plus attribué par une orientation. Nous présentons ci-dessous deux des méthodes d'approximation polygonale que nous avons implémentées pour nos besoins. Pour un bon état de l'art sur les méthodes d'approximation polygonale, le lecteur pourra se référer à [Fil 95].

5.3.1 Approximation par suivi itératif

La première méthode que nous avons étudiée est basée sur un parcours en avant du squelette, avec création d'un segment dès que l'aire entre la portion de courbe parcourue et le segment approximant celle-ci dépasse un certain seuil. L'algorithme de cette méthode, initialement proposé par Wall et Danielsson [Wal 84], est présenté [ALG. 7]. Le seuil permettant de fixer la précision de l'approximation est représenté par la variable *seuil*.

L'approximation présente l'avantage d'être rapide — un seul parcours —, peu consommatrice de place mémoire — il n'est pas nécessaire de mémoriser tous les points de la courbe — et relativement simple à mettre en œuvre. Néanmoins, la vision reste très locale au point courant lors du parcours du squelette dans la méthode originale. De ce fait, la détection des angles se fait de manière tardive et il n'y a pas préservation des points anguleux [FIG. 5.10(a)], puisqu'il n'est pas possible, lors d'un changement de direction local, de savoir s'il s'agit d'une perturbation de la courbe ou réellement d'un changement général de direction.

(a) Sans préservation des angles. (b) Avec préservation des angles.

FIG. 5.10 – *Approximation polygonale par suivi itératif.*

Cet inconvénient est partiellement résolu grâce à notre adaptation de la méthode originale [Tom 87] : lors du parcours de la chaîne, un point où la direction de la ligne change de manière notable est marqué comme point de coupure potentiel. Si la chaîne reprend plus loin la direction initiale qu'elle avait avant le marquage, la marque est effacée, car le changement de direction ne correspond alors qu'à une perturbation locale. En revanche, si ce changement de direction persiste et si le critère de Wall et Danielsson n'est plus vérifié, le segment courant est créé à partir de l'éventuel point marqué et non pas à partir du point courant [FIG. 5.10(b)].

ALG. 7 – WDApproximation::approxime(ChaînePoints ch, Réel $seuil$)

RÉSULTAT: $listeSegments$: liste des segments constituant l'approximation polygonale

1: $listeSegments \leftarrow \emptyset$
2: considérer $ch[0]$ comme origine du repère
3: $segmentCourant \leftarrow (ch[0],ch[1])$
4: $f \leftarrow 0$
5: **pour** $i \leftarrow 1$ à $(n-1)$ **faire**
6: $deltaX \leftarrow ch[i].x() - ch[i-1].x()$
7: $deltaY \leftarrow ch[i].y() - ch[i-1].y()$
8: $f \leftarrow f + deltaY \times ch[i].x() - deltaX \times ch[i].y()$
9: **si** $|f| \leq seuil \times segmentCourant$.longueur() **alors**
10: $segmentCourant \leftarrow (segmentCourant.origine(),ch[i])$
11: **si** $ch[i]$ correspond à un changement notable de direction **alors**
12: **si** il n'y a pas de point mp mémorisé **alors**
13: $mp \leftarrow ch[i]$
14: $mi \leftarrow i$
15: **fin si**
16: **sinon**
17: supprimer l'éventuelle mémorisation d'un point mp
18: **fin si**
19: **sinon**
20: **si** il y a un point mp mémorisé **alors**
21: $segmentCourant \leftarrow (segmentCourant.origine(),mp)$
22: $i \leftarrow mi$
23: supprimer la mémorisation du point mp
24: **fin si**
25: $listeSegments$.ajout($segmentCourant$)
26: considérer $ch[i-1]$ comme origine du repère
27: $segmentCourant \leftarrow (ch[i-1],ch[i])$
28: $f \leftarrow 0$
29: **fin si**
30: **fin pour**
31: $listeSegments$.ajout($segmentCourant$)

5.3.2 Approximation par découpage récursif

Nous avons implanté une deuxième méthode d'approximation polygonale. Celle-ci est basée sur les travaux de Lowe [Low 87], repris et améliorés par Rosin et West [Ros 89]. L'idée de base consiste à découper récursivement la chaîne en un ensemble de segments de plus en plus petits et précis. On

commence par le segment joignant les deux extrémités P_0 et P_{n-1} de la chaîne[21]. On cherche le point P_m de la chaîne le plus éloigné de ce segment ; soit d_m la distance de ce point au segment. On associe au segment $[P_0,P_{n-1}]$ une valeur significative $\frac{d_m}{L}$, où L est la longueur du segment. Si $d_m \approx 0$ ou si la chaîne est longue de 3 pixels ou moins, on arrête le découpage, sinon on applique récursivement le processus aux deux sous-chaînes $(P_0 \dots P_m)$ et $(P_m \dots P_{n-1})$ [ALG. 8].

ALG. 8 – RWApproximation::approxime(ChaînePoints *ch*, int *début*, int fin)

RÉSULTAT: $racine$: racine de l'arbre résultant de l'approximation polygonale

```
racine ← construitArbre(ch,début,fin)
maxDéviation ← 0
pour i ← (début + 1) à (fin − 1) faire
    d ← ch[i].distanceSegment(ch[début],ch[fin])
    si d > maxDéviation alors
        pointDeCoupe ← i
        maxDéviation ← d
    fin si
fin pour
racine.majSignificance(maxDéviation/ch[début].distancePoint(ch[fin]))
si (fin − début + 1) > 3 et maxDéviation ≥ 2 alors
    racine.majFilsGauche(approxime(ch,début,pointDeCoupe))
    racine.majFilsDroit(approxime(ch,pointDeCoupe,fin))
    rechercheMeilleureApproximation(racine)
fin si
```

À la fin de ce processus, on obtient un arbre, correspondant aux différents niveaux de découpage. Cet arbre est parcouru en partant des feuilles, en retenant à chaque fois le segment le plus significatif, c'est-à-dire celui dont la mesure significative est la plus proche de 0 [ALG. 9]. Bien que de complexité

ALG. 9 – RWApproximation::rechercheMeilleureApproximation(Racine $racine$)

```
meilleurSig ← significance minimum entre la racine et, s'ils existent, les fils gauche ou droit
si meilleurSig < racine.significance() alors
    racine.majSignificance(meilleurSig)
sinon
    suppression des fils s'ils existent
fin si
```

supérieure à la méthode par suivi itératif, cette méthode a deux avantages :

1. elle préserve beaucoup mieux les points anguleux,

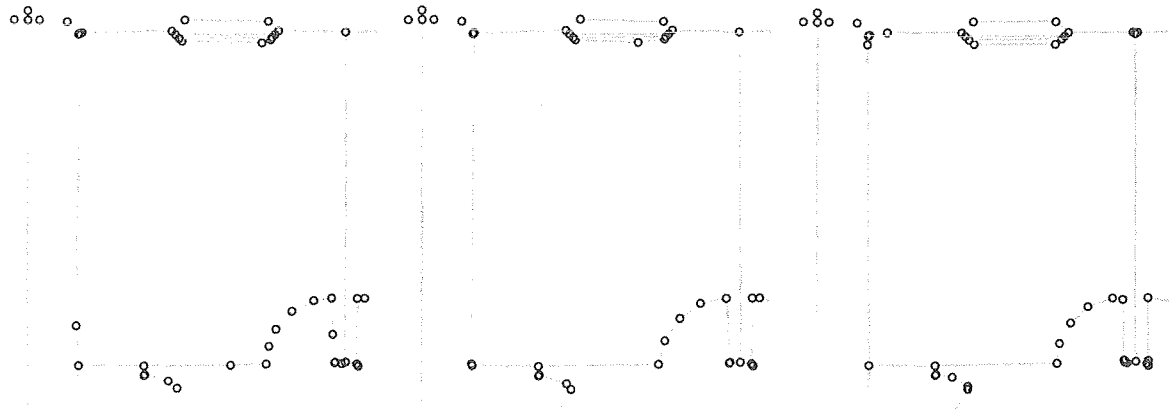
21. Dans le cas d'une boucle, on commence par découper celle-ci en deux sous-chaînes, en la coupant au point le plus éloigné du point de départ.

2. elle ne nécessite aucun seuil, puisque la valeur significative permet de pondérer automatiquement les deux critères de précision (d_m le plus faible possible) et de réduction des données (minimiser le nombre de segments en en maximisant les longueurs).

La mesure significative utilisée ici peut éventuellement être améliorée en s'inspirant de travaux plus récents de Rosin sur les mesures les plus pertinentes [Ros 97].

5.3.3 Résultats

Nous présentons ici des résultats obtenus à partir d'un squelette obtenu avec une transformée de distance 3–4, méthode de squelettisation que nous avons retenue pour nos besoins. La qualité des résultats est satisfaisante sur des images ne comportant pas de lignes trop épaisses quelle que soit l'approximation polygonale utilisée [FIG. 5.11].



(a) Suivi itératif — *seuil* = 4. (b) Suivi itératif — *seuil* = 10. (c) Découpage récursif (R&W).

FIG. 5.11 – *Primitives vectorielles résultant de la vectorisation suivant la méthode d'approximation polygonale choisie. Nombre de segments obtenus : (a) 457 segments, (b) 404 segments, (c) 474 segments (les images présentées sont des extraits).*

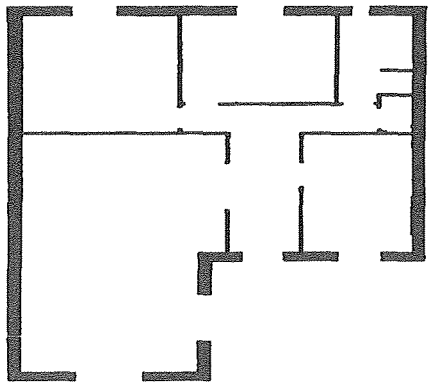
5.4 Perspectives

Notre équipe envisage actuellement plusieurs pistes pour améliorer notre processus de vectorisation. Cette étape constitue en effet une étape charnière pour la suite de l'analyse. L'extraction des primitives vectorielles approximant les lignes de l'image originale nécessite de faire un compromis entre la précision des résultats et le quantité de primitives extraites. Les primitives extraites lors de cette étape sont ensuite utilisées par toute la suite de l'analyse : il est donc essentiel d'en trouver une expression la

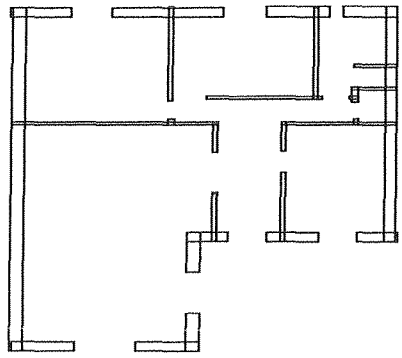
plus juste possible. Deux grandes possibilités d'évolution pour la vectorisation au sein de l'équipe sont actuellement étudiées.

- L'ajout de connaissances *a priori* par application de post-traitements sur les vectorisations de type squelettisation–chaînage–approximation polygonale [Tom 99] :
  - Röösli et Monagan [Röö 96] proposent par exemple d'ajouter des contraintes lors du processus de vectorisation pour décrire la géométrie « idéale » des résultats. Ces contraintes s'appliquent aux primitives vectorielles extraites : contraintes angulaires ou contraintes de connexion. Cela permet d'obtenir une précision accrue, moyennant cependant un coût de calcul non négligeable, le processus d'optimisation étant basé sur un recalage itératif au sens des moindres carrés. Les auteurs, qui travaillent sur des documents cadastraux, sont tenus de fournir des résultats précis et peuvent donc accepter le surcoût engendré.
  - Le même type de corrections est proposé par Chen *et al.* [Che 96]. Les auteurs ont réalisé un environnement de vectorisation évolué, dédié à l'analyse de plans mécaniques. Les résultats obtenus sont intéressants, mais reposent sur un certain nombre d'heuristiques qui introduisent des seuils et des paramètres supplémentaires, ce qui ne correspond pas à notre approche.
  - L'approche qui nous semble la plus intéressante par rapport aux choix que nous avons faits est celle de Janssen et Vossepoel [Jan 97b]. Ils proposent une méthode de vectorisation de documents techniques en plusieurs passes. Une première vectorisation du document est effectuée, en utilisant une tolérance assez grande pour l'approximation polygonale, ce qui donne une vectorisation grossière. Lors de cette vectorisation, les points remarquables (jonctions, extrémités, coins) sont recensés. Pour chacun de ces points, un élément structurant est déterminé ; sa taille et son contenu dépendent d'informations locales (épaisseur maximum des traits, application d'opérateurs morphologiques sur l'image originale). Les points remarquables sont ensuite déplacés itérativement par des convolutions grâce à l'élément structurant, jusqu'à convergence des positions. Les points remarquables sont alors appelés *points d'ancrage*. Une deuxième vectorisation basée sur l'image initiale et sur les points d'ancrage est alors effectuée pour repositionner au mieux les points remarquables. Les résultats fournis présentent une bonne localisation de ces points remarquables.
- Parallèlement à ces améliorations possibles, nous étudions actuellement d'autres voies pour obtenir un processus de vectorisation robuste et précis [Tom 00]. Notre équipe travaille actuellement sur une méthode qui propose d'aborder la vectorisation en déterminant la géométrie bidimensionnelle du document, c'est-à-dire en le décomposant en primitives graphiques 2D[22] [FIG. 5.12(b)] pour trouver les lignes [FIG. 5.12(c)] [Tab 00a]. La méthode présente certains avantages, au niveau de la précision de la localisation des points de jonction par exemple, mais reste cependant à finaliser, notamment en calculant les relations d'adjacence qui existent entre les primitives graphiques 2D.

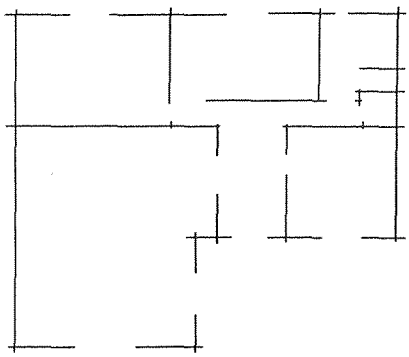
22. À ne pas confondre avec les primitives vectorielles résultant de la vectorisation.



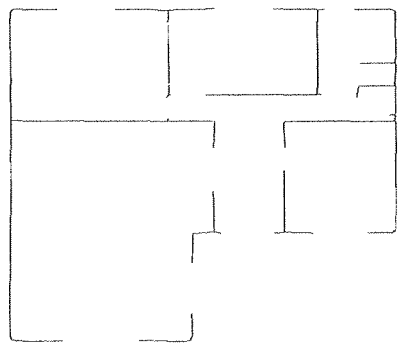
(a) Image originale.



(b) Reconstruction en primitives 2D.



(c) Axes médians.



(d) Vectorisation « classique ».

FIG. 5.12 – *Vectorisation basée sur la décomposition de l'image en primitives graphiques 2D de (a) à (c), et vectorisation de type squelettisation–chaînage–approximation (d).*

Chapitre 6

Recherche de lignes tiretées ou pointillées

6.1 Contexte et objectif

De nombreuses lignes sont représentées dans les plans architecturaux sous forme de lignes pointillées. Si ces lignes ne posent généralement pas de problème de compréhension pour l'homme, qui parvient aisément à assimiler une succession de points et de tirets alignés à une ligne pointillée, il n'en est pas de même pour l'ordinateur. Ces lignes sont en effet représentées, après la vectorisation, par une succession de primitives qui n'ont à ce stade aucun lien entre elles. Or les lignes pointillées sont généralement utilisées de la même manière dans un plan que les lignes continues : elles peuvent rentrer dans la composition d'un symbole architectural, délimiter une zone, annoter une partie du plan... Il est donc nécessaire de les identifier afin de pouvoir les utiliser dans les autres phases d'analyse. Cette détection comprend une phase d'identification des segments correspondant à des bribes de lignes pointillées et une phase de regroupement de ces segments en un nouveau segment, attribué par le style de la ligne identifiée.

Après un état de l'art des méthodes existantes [§ 6.2], nous présentons notre méthode de détection ainsi que les résultats que nous obtenons [§ 6.3].

6.2 État de l'art

Le problème de la détection de lignes tiretées ou pointillées est classique en analyse de documents techniques et a fait l'objet de nombreuses publications. Dans pratiquement tous les cas, la détection s'effectue uniquement à partir des données vectorisées. Les grandes phases algorithmiques sont relativement semblables : définition d'un modèle de ligne pointillée, construction et validation d'hypothèses de telles lignes à partir des vecteurs présents sur l'image traitée et finalement substitution des vecteurs par les lignes pointillées qu'ils décrivent.

Kasturi *et al.* proposent un algorithme [Kas 90] permettant de détecter différents types de lignes pointillées. On peut effectivement différencier les lignes pointillées ne contenant que des tirets de longueur égale, celles composées de tirets et de points, celles composées de tirets de longueurs différentes [FIG. 6.1]... Ils énoncent des contraintes génériques qu'ils associent à tout modèle de ligne pointillée :

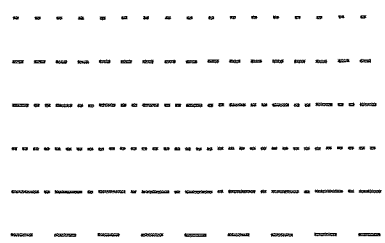


FIG. 6.1 – *Différents types de lignes tiretées ou pointillées.*

- une ligne pointillée doit comprendre au minimum quatre segments ;
- les segments entrant dans la composition d'une ligne pointillée ont une longueur maximum ;
- la longueur de l'espace séparant deux segments qui se suivent doit être comprise dans un intervalle de valeurs. Par ailleurs, cette longueur doit être constante ;
- en exprimant une ligne pointillée sous forme d'une succession de segments $(s_1, s_2, \ldots, s_n)$, la longueur d'un segment s_i doit être égale à celle du segment s_{i+2}. Cette contrainte facilite la détection des lignes pointillées présentant un motif de répétition d'au plus deux éléments.

Toutes les longueurs considérées dans les relations de comparaison ci-dessus peuvent fluctuer selon une marge pouvant aller jusqu'à 30 %. La recherche de lignes pointillées s'articule ensuite autour de deux grandes phases. La première est une étude des relations de voisinage entre les segments retenus comme candidats à partir des critères définis précédemment. Elle permet de construire une structure qui classe les voisins de chacun de ces segments sur des critères d'éloignement et d'orientation dans le plan. À partir de cette structure, des hypothèses de lignes pointillées sont générées en partant d'un segment initial. Cette hypothèse est étendue de façon incrémentale par l'ajout de deux segments à chaque fois. Outre l'avantage de pouvoir traiter plus facilement les lignes pointillées hétérogènes, cette incrémentation apporte en plus la possibilité de faire des retours en arrière. En effet, lorsque plusieurs choix de poursuite d'une hypothèse sont possibles, l'hypothèse la plus pertinente est étudiée dans un premier temps. Si cette pertinence diminue lors de la recherche du segment voisin suivant, un retour en arrière est effectué pour étudier les autres alternatives possibles au dernier choix effectué. Le critère d'orientation est utilisé de façon locale, ce qui permet de détecter des lignes pointillées qui ne sont pas forcément droites.

L'année suivante, Lai et Kasturi [Lai 91] proposent un algorithme assez similaire qui en plus de la détection effectue une classification des lignes pointillées en fonction de certains attributs : longueur moyenne des segments et des trous, détection de cycle en cas de lignes pointillées courbes, composition de ligne homogène ou hétérogène...

Boatto *et al.*, qui travaillent sur l'analyse de carte [Boa 92], utilisent une structure de graphe pour représenter l'image qu'ils analysent. Cette structure est associée à l'image de points initiale, ce qui leur permet de disposer de ces deux types de représentation pour chaque donnée lors de leurs traitements. Ils sélectionnent les tirets candidats à partir du graphe de l'image par analyse des propriétés géométriques et reconnaissance de formes typiques. Les lignes pointillées sont ensuite recherchées en se basant sur des critères d'espacement et d'alignement de tirets consécutifs. Boatto *et al.* font d'autre part apparaître

le problème des lignes pointillées qui se croisent. Dans ce cas, les intersections peuvent s'apparenter sur le plan original à des jonctions en X, T, L, etc. [FIG. 6.2] Ces jonctions ne sont pas assimilées à des

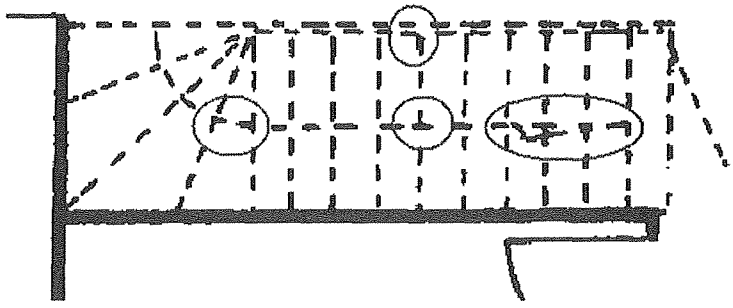


FIG. 6.2 – *Exemples de croisement de lignes pointillées, signalés par les zones encerclées.*

tirets, et ne peuvent être prises en compte dans l'élaboration d'une hypothèse de ligne pointillée. Elles sont cependant réinjectées par la suite dans les hypothèses de lignes pointillées grâce au graphe qui est associé à l'image.

Pour Vaxivière [Vax 95], les lignes pointillées contenues dans les plans mécaniques ont une signification sémantique forte. Il développe une méthode d'extraction de lignes pointillées dans le système CELLESTIN, qui s'articule autour de trois grandes étapes. Tout d'abord la localisation des segment candidats selon des critères d'alignement et d'espacement. Ensuite, l'analyse du type de ligne pointillée en fonction de sa composition. Ces lignes suivent les spécifications de la norme française NF E 04-103, ce qui permet de les classifier de façon très précise. Une fois les hypothèses générées et vérifiées, celles-ci sont remplacées par une ligne pointillée attribuée selon le type de ligne reconnue.

Joseph et Pridmore [Jos 92] adoptent une approche syntaxique dans leur système ANON pour analyser des documents techniques. À partir d'une bibliothèque de traitement d'images bas-niveau et d'une grammaire, ils développent un système flexible doté de règles stratégiques d'analyse. Leur détection de lignes tiretées s'inscrit dans ce système. Lors du suivi d'une ligne pleine pendant l'analyse, l'arrivée à une extrémité de la ligne conduit à considérer cette ligne comme le début d'une hypothèse de ligne tiretée. L'hypothèse est ensuite prolongée. Si elle s'avère effectivement être une ligne tiretée, les caractéristiques de cette ligne (composition, longueur...) sont calculées et utilisées pour extraire la ligne tiretée dans son intégralité.

Enfin, Dori *et al.* exposent dans [Dor 96b] la méthode de détection de lignes pointillées qu'ils ont utilisé pour gagner le concours de détection automatique de lignes pointillées organisé lors de la conférence GREC'95[23]. Cette méthode, qui est à la base de notre propre méthode de détection, est plus largement détaillée dans le paragraphe suivant. Les principales contraintes associées aux méthodes décrites ci-dessus sont regroupées dans le tableau [TAB. 6.1]. Les contraintes exprimées sont souvent les mêmes. Leur nature dépend fortement du type de ligne tiretées à analyser, qui varie finalement peu d'un type de document technique à l'autre.

D'autres approches plus générales, comme [Mar 97] ou [Jon 99], considèrent cette détection de lignes de façon plus générique. Pour Jonk *et al.* [Jon 99] en particulier, les approches « traditionnelles » sont limitées au niveau des résultats qu'elles génèrent, sont parfois trop sensibles au bruit et ne peuvent

23. *First International Workshop on Graphics Recognition.*

Contraintes	[Kas 90]	[Lai 91]	[Boa 92]	[Vax 95]	[Jos 92]	[Dor 96b]
Nombre minimum de segments	4				3	3
Longueur maximum des tirets	✓			✓		✓
Longueur de l'espace bornée	✓	✓		✓	✓	✓
Lignes courbes	✓	✓	✓		✓	✓
Lignes hétérogènes	✓	✓	✓	✓	✓	✓
Type de ligne déterminé		✓		✓	✓	✓
Utilisation des angles	✓	✓	✓		✓	✓
Utilisation des alignements	✓		[a]	✓	✓	✓
Retour en arrière	✓					
Gestion des intersections		✓	✓	✓[b]	✓	✓

[a] Utilisation d'une structure de graphe.
[b] En se basant sur la symétrie symbolisée par la ligne pointillée dans le contexte étudié.

TAB. 6.1 – *Étude comparative des méthodes de détection de lignes tiretées ou pointillées.*

pas être généralisées à des types de lignes pointillées plus complexes. Sa définition des lignes pointillées s'exprime par :

- une ligne centrale qui « porte » la ligne pointillée ;
- un ensemble de symboles, ou motifs, rentrant dans la composition de la ligne pointillée ;
- une grammaire qui définie la répétition des motifs selon la ligne centrale.

À partir de cette définition, il peut alors extraire de nouveaux types de lignes pointillées, comme ceux présentés [FIG. 6.3]. Sa détection s'articule alors autour de plusieurs étapes : après la détection des différents symboles et la détermination de la ligne centrale, Jonk met en œuvre une méthode qui lui permet d'inférer la grammaire de la ligne pointillée qu'il analyse. À partir de cette grammaire et des hypothèses qu'il génère, sous forme de chaînes de symboles, il calcule la distance qui sépare la chaîne de la grammaire. Cette distance est calculée à partir des opérations classiques de manipulation des chaînes, telles que l'insertion, la substitution ou encore la suppression.

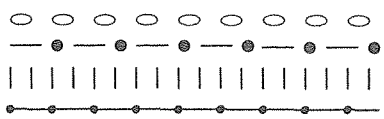


FIG. 6.3 – *Des lignes pointillées composées d'un plus vaste choix de symboles [Jon 99].*

6.3 Présentation de notre méthode

Les plans architecturaux que nous étudions comportent parfois des lignes pointillées. Nous avons donc développé un algorithme de détection sur ces lignes, adapté à nos besoins. En particulier, notre détection est effectuée avant la détection d'arcs et s'oriente donc exclusivement vers la détection de lignes pointillées droites. Nous devons d'autre part conjuguer avec la qualité des dessins que nous traitons, qui ne sont pas des documents normalisés, et sur lesquels on trouve un grand éventail de lignes pointillées différentes. En particulier, les dessins ne respectent pas toujours les critères de régularité (longueur des

tirets, des espaces) de par la réalisation du concepteur ou à cause du bruit lié à la numérisation du dessin. Notre stratégie ne tient donc pas compte de ces renseignements qui, s'ils peuvent s'avérer fiables et utiles dans certaines situations, peuvent induire le système d'analyse en erreur. Nous n'utilisons pas d'autre part le type de ligne pointillée détectée : tiretée, pointillée, ou les deux à la fois. Notre but est simplement de détecter ces lignes et de les substituer par des lignes continues qui nous permettrons de continuer notre analyse haut-niveau.

Notre méthode repose sur l'approche de Dori [Dor 96b]. Nous introduisons ci-après la terminologie utilisée en nous basant sur ses travaux :

- un *segment* est une primitive extraite de l'image de points grâce à la vectorisation. Ce segment est attribué par son épaisseur ;
- un *tiret* est une partie d'une ligne pointillée de l'image initiale. Il correspond dans notre cas à un segment issu de la vectorisation ;
- un *point* est une partie d'une ligne pointillée, toujours de l'image initiale. Il est présent dans les lignes composées alternativement de tirets et de points. Ce point présent est généralement représenté par un petit segment (2 ou 3 pixels) après la vectorisation, ce qui est dû à la résolution de numérisation, 300 dpi au minimum dans notre cas ;
- l'extrémité d'un segment est dite *libre* si elle n'est connectée à aucune autre primitive. Par extension, un segment est dit *libre* s'il possède au moins une extrémité libre ;
- une *clé* est un segment libre, potentiellement un tiret ou un point, qui respecte les contraintes, décrites ci-après, associées à ce type d'entités ;
- un *trou* correspond à la zone comprise entre deux extrémités de deux clés voisines appartenant à la même ligne pointillée ;
- l'*angle de déviation* est la valeur absolue de la différence d'orientation dans le plan de l'hypothèse de ligne pointillée courante par rapport à la clé courante ;
- une *ligne virtuelle* est une ligne imaginaire supportant les clés consécutives qui composent une ligne pointillée.

Nous introduisons également des contraintes propres aux types de lignes pointillées que nous recherchons dans les plans architecturaux :

- les clés ont une longueur maximum $L_{\max}$;
- les clés-tirets ont une longueur minimum $L_{\min}$. Si la longueur d'une clé est inférieure à $L_{\min}$, cette clé est considérée comme une clé-point ;
- les lignes pointillées sont composées d'au moins 3 clés. Les clés-tirets d'une même ligne pointillée ont approximativement la même longueur. Il en est de même par définition pour les clé-points ;
- les clés sont séparées par des trous de longueur régulière ;
- les clés sont alignées selon une ligne virtuelle.

Notre détection commence par l'extraction des clés de l'image vectorisée que nous analysons à partir des segments qu'elle contient. Cette extraction se fait sur les critères de longueur définis ci-dessus, c'est-à-dire par comparaison de la longueur des segments de l'image par rapport aux paramètres $L_{\min}$ et $L_{\max}$. Nous générons ensuite des hypothèses de lignes pointillées à partir des clés extraites en suivant l'algorithme [ALG. 10].

Pour commencer une hypothèse, nous choisissons une clé libre inutilisée, point de départ d'une nouvelle hypothèse de ligne pointillée, et nous essayons de l'étendre dans les deux directions afin de trouver d'autres segments décrivant cette ligne. Cette hypothèse est complétée tant qu'il est possible

ALG. 10 – Image::DetecteLignesPointillées()

RÉSULTAT: $listeLignesPointillées$
1: $listeLignesPointillées \leftarrow \emptyset$
2: **pour** chaque clé c **faire**
3: **si** c **non** utilisé **et** c est un tiret **alors**
4: $lignePointillée \leftarrow \emptyset$
5: c marqué comme utilisé
6: $lignePointillée$.ajout(c)
7: $sens \leftarrow$ TrouveMeilleurSens(c)
8: $lignePointillée \leftarrow$ ÉtendHypothèse(c, $sens$, $lignePointillée$)
9: $lignePointillée \leftarrow$ ÉtendHypothèse(c, $\neg sens$, $lignePointillée$)
10: **si** $lignePointillée$.Taille() ≥ 3 **alors**
11: $listeLignesPointillées.Ajout(lignePointillée)$
12: **sinon**
13: les clés de $lignePointillée$ sont marquées **non** utilisées
14: **fin si**
15: **fin si**
16: **fin pour**

de trouver des nouveaux segments à ajouter. Pour que cette étape soit efficace, il est nécessaire d'avoir l'hypothèse la plus pertinente possible. Pour cela, nous effectuons avant le début de la recherche une étude des voisins les plus proches de la clé c. Nous commençons ensuite la recherche dans le sens où le meilleur score est obtenu, ce qui permet de démarrer avec une hypothèse plus robuste. Cette recherche est décrite par l'algorithme [ALG. 11].

Pour chercher des segments permettant de compléter cette hypothèse, nous déterminons une zone de recherche, adjacente à l'extrémité désignée par le sens de recherche, dont la largeur est le double de la largeur de la clé courante, et dont la longueur est égale à la distance maximale autorisée entre deux segments [FIG. 6.4]. Nous extrayons alors les clés qui se trouvent dans l'aire de recherche. La fonction

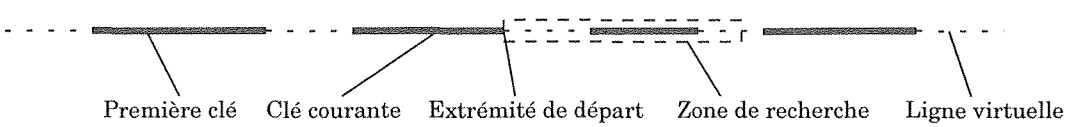


FIG. 6.4 – *Zone de recherche lors de l'extraction de ligne pointillées.*

`ChercheSuivant` effectue ensuite un tri sur les différentes clés trouvées dans la zone de recherche. Ce tri est effectué à l'aide d'une fonction de calcul de distance `Distance` dont l'algorithme est fourni [ALG. 12] (avec $\Delta(x,y)$ définissant la différence d'angle entre deux segments x et y). Les différentes constantes utilisées dans cet algorithme sont :

- `MINLONGTIRET` : longueur minimale d'une clé pour être considérée comme un tiret. En dessous de cette longueur, la clé est considérée comme un pointillé. La longueur est *a priori* fixée en fonction

ALG. 11 – Image::ÉtendHypothèse(Clé *clé*, Sens *sens*, liste<Clé> *lignePointillée*)

RÉSULTAT: *lignePointillée*
1: *stop* ← **faux**
2: **tant que non** *stop* **faire**
3: *zone* ← ConstruitZone(*clé*, *sens*)
4: *suivant* ← ChercheSuivant(*clé*, *zone*, *lignePointillée*)
5: **si** *suivant* existe **alors**
6: *suivant* est marqué comme utilisé
7: *lignePointillée*.Ajout(*suivant*)
8: *clé* ← *suivant*
9: **sinon**
10: *stop* ← **vrai**
11: **fin si**
12: **fin tant que**

ALG. 12 – Clé::Distance(Clé *départ*, liste<Clé> *lignePointillée*)

RÉSULTAT: *distance*
1: $distanceEucli$ ← distance Euclidienne entre *moi* et la dernière clé de *lignePointillée*
2: $ligneVirtuelle$ ← ligne joignant les centres de la première et la dernière clé de *lignePointillée*
3: $diffAngle$ ← Δ(*moi*, $ligneVirtuelle$)
4: $ligneVirtuelle$ ← ligne joignant le centre de la première clé de *lignePointillée* et mon centre
5: $écartMax$ ← 0
6: **pour** chaque clé *c* de *lignePointillé* **faire**
7: *écart* ← distance Euclidienne entre le centre de *c* et $ligneVirtuelle$
8: **si** $écart > écartMax$ **alors**
9: $écartMax$ ← *écart*
10: **fin si**
11: **fin pour**
12: **si** ((Longueur() > `MINLONGTIRET` **et** $diffAngle$ > `MAXDÉVIATION`) **ou**
($écartMax$ > `MAXÉCART`) **ou**
j'ai une extrémité commune à la dernière clé de *lignePointillée*) **alors**
13: *distance* ← ∞
14: **sinon**
15: *distance* ← $distanceEucli$
16: **fin si**

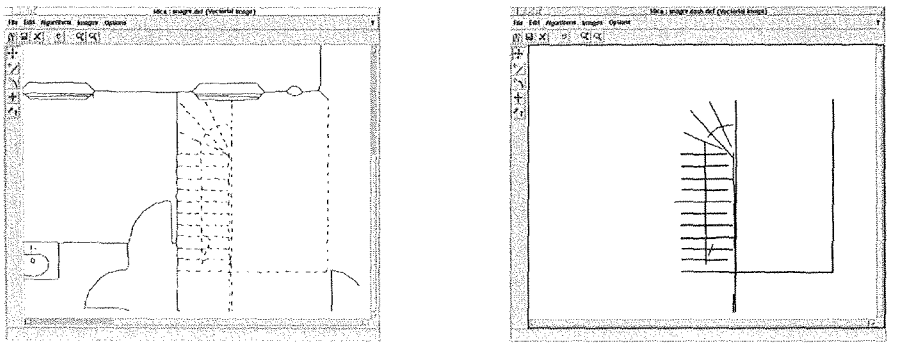
de la résolution de numérisation. Dans notre cas, pour une résolution de 300 dpi, nous avons opté pour une valeur de `MINLONGTIRET` à 4.

- `MAXDÉVIATION` : différence angulaire maximale autorisée entre l'orientation de la clé courante et l'orientation de la ligne virtuelle. Cette contrainte nous permet de ne retenir que les clés dont l'orientation est sensiblement la même que celle de l'hypothèse de ligne pointillée actuelle. Elle n'est en fait utilisée que si la clé est un tiret (nous ne considérons pas pertinente l'orientation d'un pointillé) et que si l'hypothèse actuelle de ligne pointillée comporte au moins 2 clés. Nous avons fixé expérimentalement cette valeur à 15.
- `MAXÉCART` : distance maximum entre la nouvelle ligne virtuelle — incluant la nouvelle clé — et le centre de chaque clé de cette ligne virtuelle. Cette constante contraint la recherche de ligne pointillée droite. Rappelons que dans notre cadre, nous effectuons la recherche de lignes pointillées avant la recherche d'arcs de cercle. Cette contrainte, très forte dans notre cas, a été fixée expérimentalement à 4.

Cette opération étant très coûteuse en temps de traitement, nous proposons une approche basée sur l'utilisation d'une structure de baquets [Asa 85] (voir aussi annexe B) qui permet un découpage de l'image en un ensemble de cases de mêmes dimensions. L'utilisation de ces baquets accélère la recherche des segments contenus dans une zone de l'image. La longueur et la largeur des baquets sont égales à la distance maximum autorisée entre deux segments supportés par une ligne pointillée.

La nouvelle clé doit être alignée avec les clés déjà détectées, ou avoir une longueur inférieure à la longueur minimale autorisée pour les tirets. Dans ce cas, la clé n'est pas un tiret, mais un point, et nous vérifions uniquement qu'elle se trouve entièrement dans la zone de recherche pour la considérer comme valide. Si l'une de ces contraintes est vérifiée, la nouvelle clé est ajoutée à l'hypothèse de ligne pointillée courante. Nous continuons la recherche de nouvelles clés à ajouter en considérant les clés se trouvant aux extrémités de l'hypothèse, jusqu'à ce qu'il ne soit plus possible d'étendre l'hypothèse courante dans aucune direction.

Une hypothèse de ligne pointillée comportant au moins 3 clés est gardée : les clés correspondant à l'hypothèse sont supprimées et un segment correspondant à la ligne pointillée décrite par les clés est créé. Dans le cas contraire, l'hypothèse est rejetée. La boucle principale continue tant qu'il reste des clés. Les résultats obtenus sont présentés [FIG. 6.5].



(a) Image source. (b) Lignes pointillées détectées.

FIG. 6.5 – *Détection de lignes pointillées sous MICA.*

Certaines améliorations ont été apportées à la phase de détection de lignes pointillées. En effet, dans de nombreux cas, les lignes ainsi détectées constituent les données d'un nouveau traitement, comme dans

le cas de la détection de textures. Pour cette dernière, basée sur la recherche et sur l'étude des polygones présents dans le plan, l'importance de la connexité des segments est capitale.

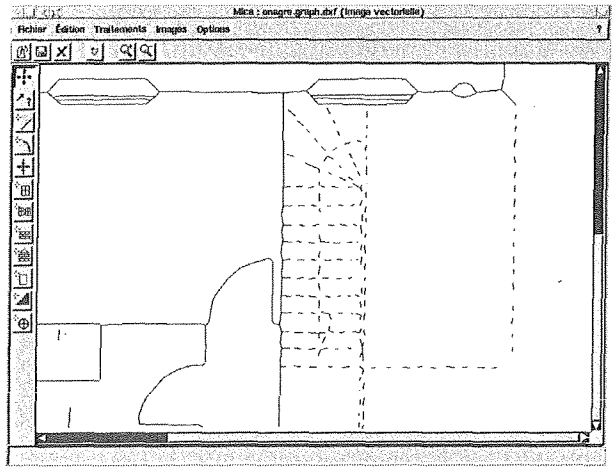
Or, dans de nombreux cas, les lignes pointillées détectées ne sont pas raccordées aux segments existants. C'est en particulier le cas lorsque le premier ou le dernier tiret composant une ligne pointillée n'est pas lui-même connexe à un autre segment. Nous avons donc complété la méthode en lui adjoignant un post-traitement qui essaie de prolonger les lignes pointillées détectées pour tenter de les raccorder aux segments les avoisinant lorsque cela est possible. L'algorithme de ce post-traitement est présenté [ALG. 13]. L'écart maximum entre l'extrémité d'une ligne pointillée et une ligne voisine ne peut

ALG. 13 – Image::AjusteHypothèse(Segment $segmentPointillé$, Sens $sens$)

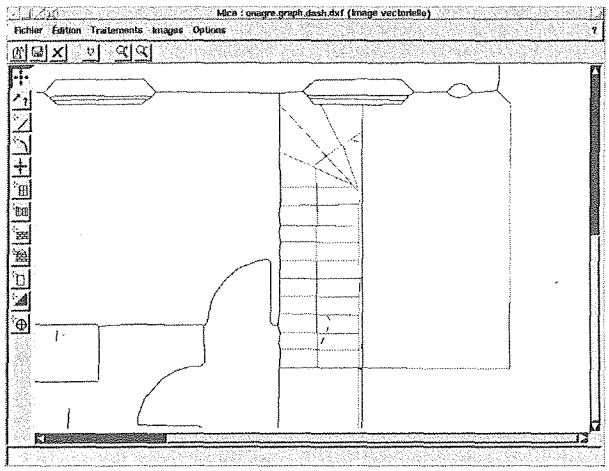
RÉSULTAT: $nouveauSegment \leftarrow segmentPointillé$

1: **si** l'extrémité de $segmentPointillé$ dans le sens $sens$ est libre **alors**
2: $meilleureDistance \leftarrow -\infty$
3: $lignePointillée \leftarrow$ Ligne($segmentPointillé$)
4: $listeVoisins \leftarrow$ RechercheVoisins($segmentPointillée$, $sens$)
5: **pour** chaque segment voisin v de $listeVoisins$ **faire**
6: **si** $lignePointillée$ et v ont un point p d'intersection **alors**
7: $distance \leftarrow$ distance Euclidienne entre p et $segmentPointillé$
8: **si** (($meilleureDistance < 0$ **et** $distance > meilleureDistance$) **ou** ($meilleureDistance > 0$ **et** $distance < meilleureDistance$)) **alors**
9: $meilleureDistance \leftarrow distance$
10: $nouveauSegment \leftarrow segmentPointillé$ avec p comme nouvelle extrémité dans le sens $sens$
11: **fin si**
12: **fin si**
13: **fin pour**
14: **si** $meilleureDistance = -\infty$ **alors**
15: Même itération que ci-dessus (5–13) en convertissant préalablement les segments v en lignes
16: **fin si**
17: **fin si**

excéder la longueur maximum autorisée entre deux tirets de la ligne pointillée. Nous obtenons à l'issue de ce post-traitement de meilleurs résultats au niveau de la connexité [FIG. 6.6] et [FIG. 6.7].

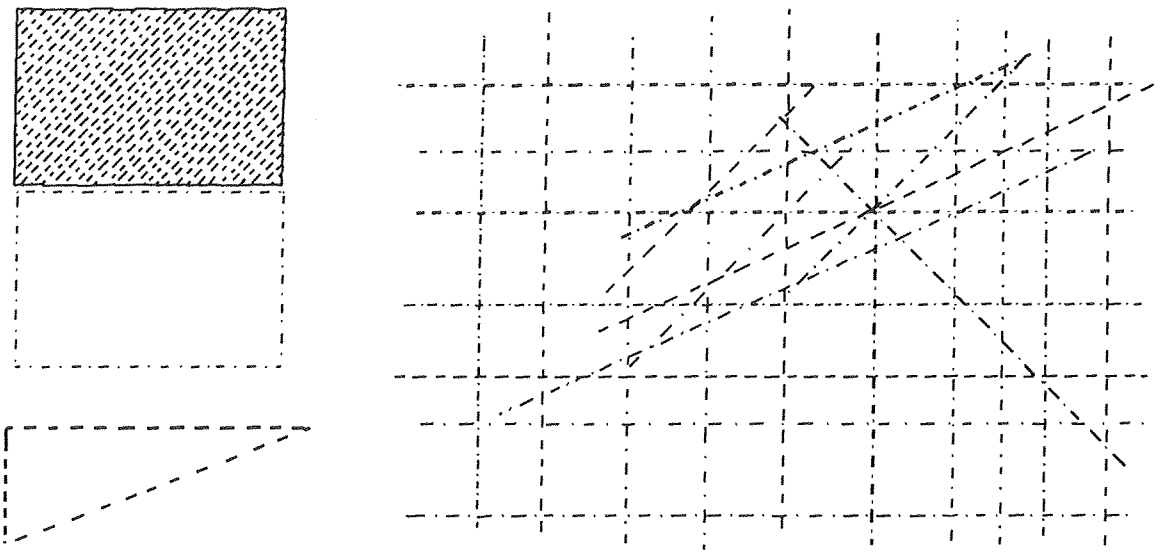


(a) Plan de départ.

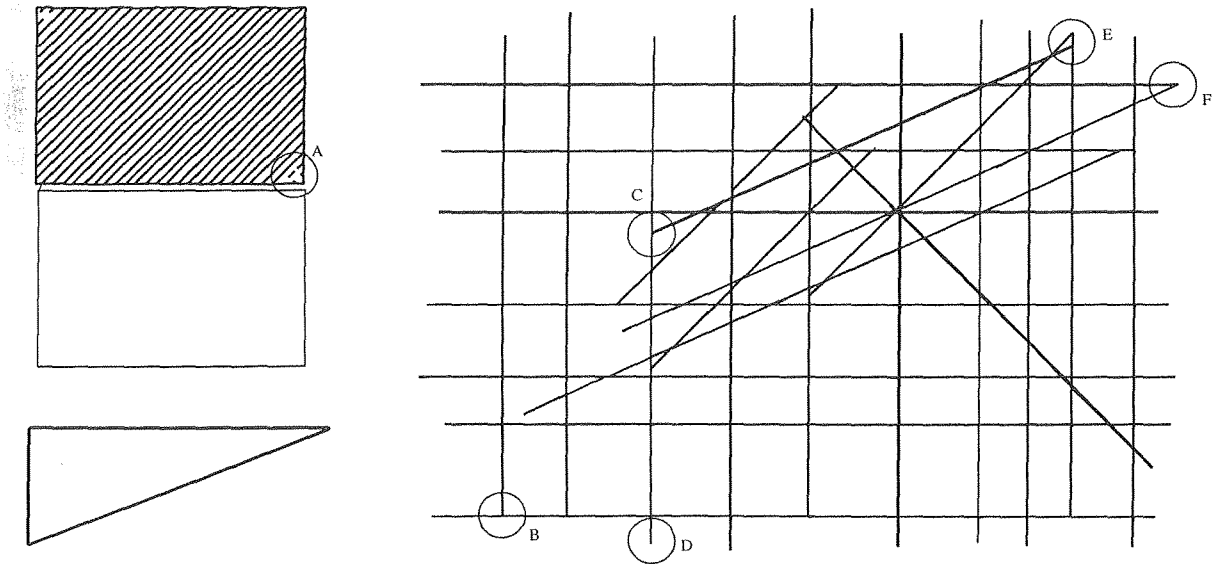


(b) Lignes pointillées détectées.

FIG. 6.6 – *Lignes pointillées détectées et ajustées sous MICA.*



(a) Image vectorisée de départ.



(b) Lignes tiretées et pointillées détectées.

FIG. 6.7 – *Détection de lignes tiretées et pointillées sur un exemple utilisé lors du concours de vectorisation de la conférence* GREC'95. *Quelques annotations : A) ligne non reconnue en raison d'un nombre insuffisant de clés B) ligne ajustée C) ligne ajustée D) ligne non ajustée car trop longue E) ajustement importun — les lignes concernées n'ont pas d'extrémité commune à l'issue de la détection et ont été prolongées F) lignes ajustées.*

Chapitre 7

Détection des arcs de cercle

7.1 Introduction

Suite aux étapes de vectorisation (chapitre 5) et de détection de lignes pointillées (chapitre 6), nous disposons maintenant d'une description de l'image à analyser sous forme de segments, attribués par un style de dessin (continu ou pointillé) et une épaisseur. Si cette représentation est nettement plus symbolique que celle des images numérisées, l'unique type de primitives graphiques disponibles (les segments) ne permet pas d'exprimer toute la variété de formes utilisée dans un document technique. Sur les plans architecturaux en particulier, toutes les courbures qui se trouvent au niveau des portes et des fenêtres sont pour le moment décomposées en un ensemble de segments, ce qui ne constitue pas une représentation très pertinente de ce type d'éléments.

En fait, dans la majeure partie des systèmes d'analyse de documents techniques, la vectorisation comprend une étape de recherche des arcs de cercle présents dans l'image. Cela permet d'augmenter le vocabulaire de description de l'image d'une nouvelle primitive vectorielle, et de faciliter ainsi les traitements de haut-niveau qui interviennent dans la suite du processus d'analyse, comme la détection de symboles. Cette étape n'est pas toujours différenciée dans la littérature du processus même de vectorisation, certaines méthodes permettant d'extraire lors du même traitement les segments et les arcs de cercle contenus dans une image numérisée. Cependant, afin d'être le plus générique possible et également de répondre à nos besoins, nous avons différencié ces deux étapes. Cela nous permet, par exemple, de pouvoir détecter des arcs de cercle pointillés à ce stade, ce qui n'aurait pas été possible autrement.

Les méthodes permettant d'extraire ce type de primitives peuvent être décomposés en deux grandes catégories [Dor 95b, Dor 98b] :

- les méthodes à base de transformée de Hough, ce qui est chronologiquement la première approche envisagée pour ce traitement. Cette catégorie de méthodes travaille directement à partir de l'image numérisée, en réduisant la recherche d'arcs à celle de points d'accumulations dans un espace de paramètres. Ces méthodes fournissent généralement de bons résultats, sont robustes au bruit, mais

sont très coûteuses en temps de calcul et en espace mémoire car elles travaillent sur l'image numérisée, et donc sur la totalité des points contenus dans une telle image. En outre, dans notre cas, elles ne permettent pas de s'affranchir du problème des arcs de cercle pointillés. Le lecteur pourra se reporter à [Ill 88] pour un état de l'art sur la transformée de Hough et à [Con 88] ou à [Har 92b] pour des applications sur la détection d'arcs ;

- les méthodes basées sur l'estimation de la courbure de lignes. Cette catégorie de méthodes ne travaille pas sur l'image entière, mais uniquement sur des lignes extraites de cette image. Des méthodes ont été proposées pour travailler soit directement sur les lignes, représentées par des chaînes de points, soit sur une approximation polygonale de ces lignes.

Dans notre contexte, nous nous sommes intéressés à la seconde famille d'approches. Elle permet d'extraire plus rapidement les arcs de cercle, de considérer les arcs de cercle pointillés[24], et est également compatible avec le tuilage : les arcs peuvent être recherchés après la fusion des tuiles. En utilisant une autre approche, la méthode devrait potentiellement être capable d'effectuer la détection d'arcs dont la représentation se situe sur plusieurs tuiles, ce qui serait beaucoup plus complexe à mettre en œuvre.

Après une présentation succincte de quelques méthodes procédant à l'estimation de la courbure des lignes [§ 7.2], nous présentons la méthode que nous avons implémentée dans la plate-forme ISADORA [§ 7.3]. Nous concluons par les résultats obtenus et quelques perspectives [§ 7.4].

7.2 Quelques méthodes à la base de notre approche

Les méthodes étudiant la courbure des lignes d'une image nécessitent un traitement préalable, permettant d'extraire les lignes de l'image étudiée. Ces lignes peuvent être extraites grâce à une détection de contours, ou à partir de l'une des méthodes proposées [§ 5.2]. L'objectif est à chaque fois de se doter de lignes d'une épaisseur d'un point. La détection d'arcs peut alors être directement réalisée sur ces lignes, ou sur une approximation polygonale de celles-ci.

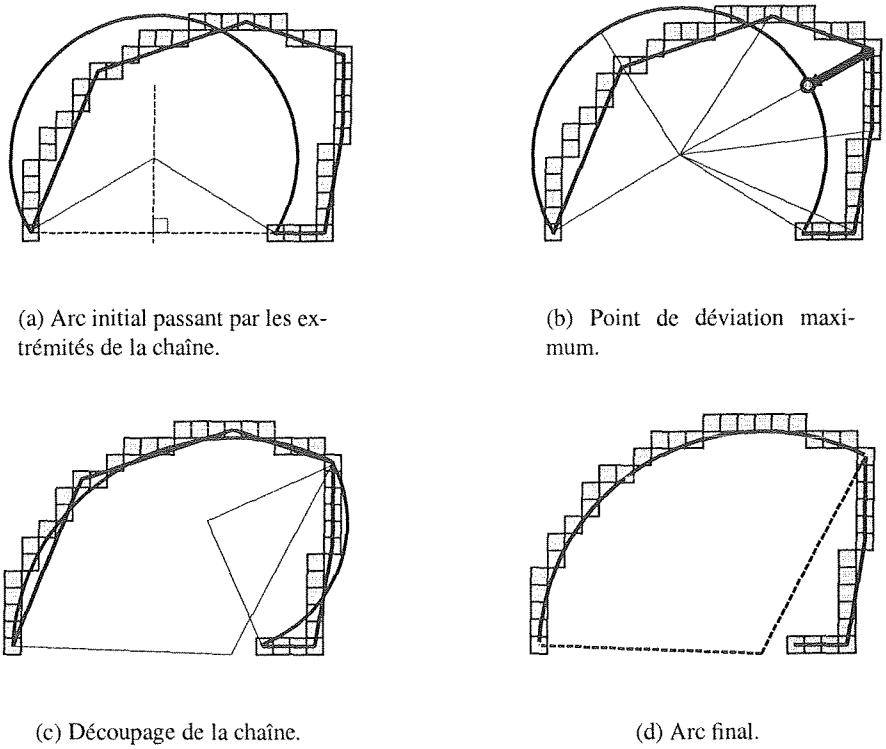
Dans la famille de méthodes dont l'entrée est constituée des chaînes de points représentant les lignes de l'image, Rosin et West [Ros 89] ont proposé une méthode basée sur un découpage récursif. Cette méthode, qui ne nécessite aucun seuil explicite, segmente une ligne en un ensemble d'arcs et de segments. Elle a déjà été présentée en partie lors de la vectorisation [§ 5.3] et se base sur les travaux de Lowe [Low 87]. Pour segmenter l'image en primitives vectorielles, la méthode repose sur la détermination de points de courbure maximum, calculés à partir du ratio entre la déviation maximum et la longueur des segments approximant la chaîne. Ce ratio est appelé *mesure de significance*. Lowe utilisait initialement cette mesure pour approximer une chaîne de points par un ensemble de segments. Rosin et West ont amélioré cette méthode en ajoutant à l'issue de cette première étape une détection d'arcs de cercle, où les arcs sont substitués aux segments lorsqu'ils représentent une meilleure approximation de la chaîne initiale.

La reconnaissance d'arcs nécessite cependant plus de règles que la reconnaissance de segments, et la seule mesure de significance n'est pas suffisante pour caractériser un arc. Par conséquent, Rosin et West ont fixé des contraintes supplémentaires :

- la connexité : les extrémités des arcs doivent obligatoirement être sélectionnées parmi les extrémités des segments approximant la chaîne de points,
- l'arité : l'arc doit contenir au minimum 4 points, étant donné qu'il est toujours possible de faire passer un arc par trois points, ou moins, non colinéaires,

24. Cette fonctionnalité n'est cependant pas prise en compte actuellement, voir [§ 7.3].

– la géométrie : le centre de l'arc doit être équidistant des deux extrémités de cet arc, ce qui contraint sa position sur la médiatrice du segment défini par les deux extrémités. La position exacte est alors calculée par une simple minimisation au sens des moindres de la somme des distances entre l'arc et les extrémités des segments issus de l'approximation polygonale [FIG. 7.1].



(a) Arc initial passant par les extrémités de la chaîne.

(b) Point de déviation maximum.

(c) Découpage de la chaîne.

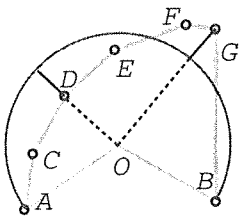
(d) Arc final.

FIG. 7.1 – *Principe de la méthode de Rosin et West.*

Afin d'améliorer cette méthode, la mesure de significance utilisée dans cette version a ensuite été remplacée par une mesure plus évoluée [Ros 97]. La méthode est intéressante, mais comporte certaines limitations. En particulier, la chaîne de points initiale est toujours coupée au point de déviation maximum, et la détection d'arcs est ensuite relancée sur chaque sous-chaîne. Dans certains cas, ce point de déviation maximum n'est pas le point le plus pertinent [FIG. 7.2], et le découpage correspondant ne permet pas d'obtenir une détection correcte. De plus, le calcul de l'erreur d'approximation d'une chaîne par un arc est uniquement réalisé sur les extrémités des segments issus de l'approximation polygonale de cette chaîne ; l'arc retenu est donc celui qui est le plus proche de l'approximation polygonale, et non pas de la chaîne de points initiale.

Dans la famille de méthodes complètement basées sur des données vectorisées, Dori a proposé une méthode appelée PBT[25] [Dor 95b]. Une vectorisation du document, non-systématique (OZZ [§ 5.2.3]), permet d'extraire des *barres* qui, lorsqu'elles sont connexes, sont regroupées pour former des *polylignes*. Des seuils sont utilisés pour sélectionner les barres qui ne sont ni trop longues, ni trop courtes, et qui présentent deux à deux la même différence angulaire. Ces seuils permettent de retenir les polylignes (partielles ou complètes) qui constituent les hypothèses d'arcs. La méthode est ensuite basée sur le fait que trois points différents non colinéaires permettent de déterminer un cercle. Les deux premiers points

25. *Perpendicular Bisector Tracing.*



d(C) = 13.0087
d(D) = 18.8197
d(E) = 8.15986
d(F) = 12.0387
d(G) = 18.2674

FIG. 7.2 – *Exemple de découpage inopportun au niveau d'un point de déviation maximum. Les deux sous-listes produites sont ensuite analysées séparément, ce qui ne permet plus de détecter l'arc existant.*

considérés correspondent aux extrémités de la polyligne retenue, et le troisième est calculé comme l'intersection de la bissectrice portée par les deux premiers avec l'hypothèse d'arc. Le centre de ce cercle est ensuite calculé comme le point d'intersection des trois bissectrices perpendiculaires du triangle formé par les trois points. Une première position du centre de l'arc peut alors être calculée. Cette position est ensuite affinée en utilisant les sommets de la polyligne considérée dans l'hypothèse d'arc, ce qui permet de déterminer une zone dans laquelle se trouve potentiellement le centre de l'arc. Chaque point de cette zone est alors testé pour déterminer la meilleure position du centre de l'arc, au sens des moindres carrés [FIG. 7.3].

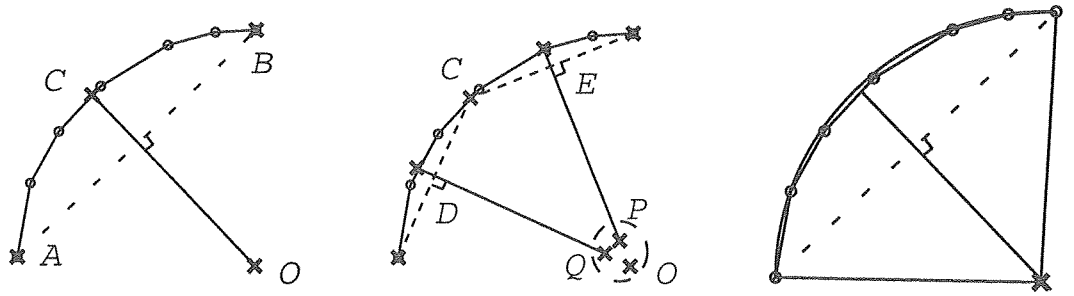


FIG. 7.3 – *Principe de la méthode de Dori pour le calcul du centre d'un arc.*

Dori a repris ce principe plus tard avec Liu [Dor 98b, Dor 99] pour proposer une nouvelle méthode itérative appelée SRAS[26]. Après la validation de l'hypothèse d'arc initiale, la méthode essaie d'étendre incrémentalement l'arc à ses extrémités. Ces tentatives d'extension sont réalisées à l'aide de zones possibles d'extension et en retestant l'hypothèse d'arc, en termes de largeur de segments, de mesure angulaire et de connexité. Pour chaque extension réalisée, le centre du nouvel arc est recalculé selon la même méthode. La méthode fournit de bons résultats, mais est limitée par les seuils sur lesquels elle repose, et sur la convergence du calcul du centre de l'arc qui considère uniquement les extrémités des barres.

Ainsi, les méthodes présentées dans ce paragraphe sont limitées par le type de données qu'elles considèrent. En effet, le calcul de l'erreur associée à une hypothèse d'arc est toujours réalisé en considérant les extrémités des segments approximant les lignes de l'image. Même si l'utilisation des données fournies par l'approximation polygonale est utile pour déterminer de manière assez simple les hypothèses d'arcs, l'ajustement de ces hypothèses n'est pas très précis, les données servant de référence n'étant qu'une approximation de la forme originale.

26. *Stepwise Recovery Arc Segmentation Algorithm.*

Partant de ces constations, nous proposons une approche hybride pour résoudre le problème de la détection d'arcs : la construction des hypothèses d'arcs basée sur le résultat de l'approximation polygonale, et l'ajustement de ces hypothèses par confrontation à la chaîne de points originale lorsque cela est possible. Notre objectif est d'obtenir un ajustement de l'hypothèse plus précis que dans les méthodes présentées ci-dessus, les chaînes de points étant plus fidèles à la forme originale que leur approximation polygonale.

7.3 Recalage des hypothèses d'arcs

La méthode que nous utilisons [Dos 00b] est basée sur les travaux de Rosin et West, à laquelle nous avons ajouté des idées de la méthode de Dori et également quelques améliorations. La plus importante d'entre elles réside dans le calcul de l'erreur associée à chaque hypothèse d'arc, non pas en nous basant sur les segments fournis par l'approximation polygonale, mais sur la chaîne de points initiale. Pour cela, nous avons adapté notre étape de vectorisation :

- en plus des données vectorisées, nos modules de vectorisation (correspondant aux différentes combinaisons des étapes présentées chapitre 5) fournissent les chaînes de points issues de la recherche des lignes ;
- les données vectorisées résultantes sont représentées non pas par des segments, mais par des *polylignes*, c'est-à-dire des suites de segments connexes entre eux. Dans ces conditions, chaque chaîne de points issue de la recherche des lignes peut être approximée par une et une seule polyligne, ce qui facilite la correspondance entre les deux structures de données.

Afin de pouvoir aisément passer d'une représentation à l'autre, nous faisons correspondre chaque chaîne de points à la polyligne qui l'approxime par une structure de données *ad hoc* au début du processus de détection d'arcs. Un simple lien permet alors de passer d'une représentation à l'autre. Cette structure ne peut cependant être valable que pour les polylignes continues, la chaîne de points correspondant aux lignes tiretées n'étant que partiellement disponible. Au niveau de la méthode, notre détection d'arcs est décomposée en deux grandes phases : la génération d'hypothèses d'arc par filtrage, et la validation de ces hypothèses [FIG. 7.4].

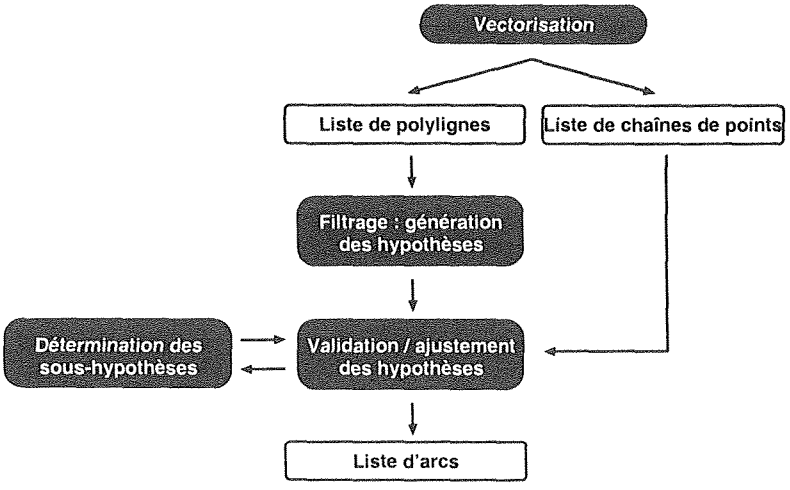


FIG. 7.4 – *Schéma général de la détection d'arcs.*

Les hypothèses sont construites à partir des polylignes fournies par l'approximation polygonale. À partir d'une polyligne, nous filtrons la liste des segments $(S_1, \ldots, S_n)$ contenus, définis par leurs extrémités $(P_1, \ldots, P_{n+1})$, afin que :

- la ou les portions de la polyligne originale contiennent un nombre minimum de points. Nous considérons que quatre points sont nécessaires pour construire une hypothèse d'arcs pertinente, étant donné qu'il est toujours possible de faire passer un arc par trois points,
- les angles successifs $\widehat{S_i\, S_{i+1}}$ et $\widehat{S_{i+1}\, S_{i+2}}$ soient similaires à un degré de liberté près.

À ce stade, chaque portion de polyligne résultante constitue une hypothèse d'arc, qui sera validée ou non dans la suite de la détection. Si un arc n'est pas détecté pour une hypothèse donnée, nous réduisons le nombre de segments de cette hypothèse, et nous la testons à nouveau, jusqu'à l'obtention d'un arc valide ou jusqu'à ce qu'il n'y ait plus assez de points pour construire une hypothèse consistante.

La phase de validation des hypothèses est réalisée en utilisant l'approche de minimisation au sens des moindres carrés proposée par Rosin et West. La mesure de l'erreur est alors effectuée, non pas sur l'approximation polygonale qui a permis de construire l'hypothèse, mais sur la chaîne de points correspondante, accessible facilement grâce à notre structure d'indexation. Si la mesure d'erreur est supérieure à une valeur maximale autorisée, l'hypothèse d'arc n'est pas validée et la méthode est appliquée aux sous-hypothèses consistantes calculables à partir de l'hypothèse initiale. Dans le cas contraire, l'hypothèse d'arc est validée et le centre de l'arc de cercle est ainsi positionné plus précisément que par recalage sur les extrémités des segments de l'approximation. L'épaisseur de l'arc détecté correspond alors à l'épaisseur de la polyligne dont il est issu.

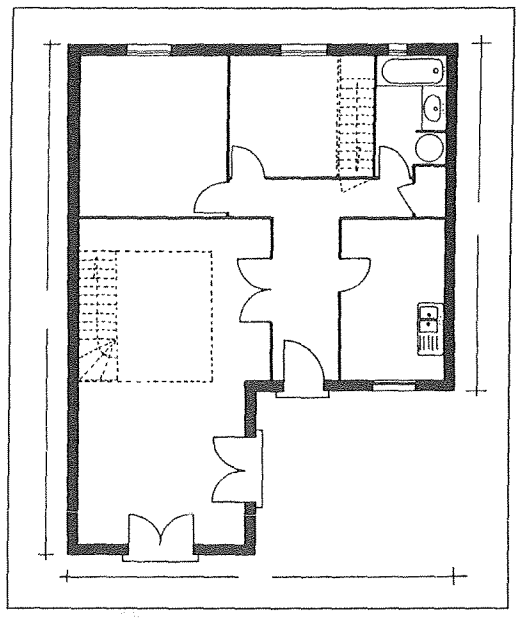
Notre méthode permet également de détecter des cercles complets. Lorsque nous travaillons sur une polyligne décrivant une boucle, nous éliminons un des segments, et nous appliquons la méthode précédemment décrite. Si un unique arc est détecté sur la polyligne résultante, nous testons la présence d'un cercle en vérifiant la validité du dernier segment, obtenu en joignant le premier et le dernier segment de cette polyligne. Dans le cas contraire, nous éliminons un autre segment de la polyligne originale et nous réappliquons la détection d'arcs.

7.4 Résultats et conclusion

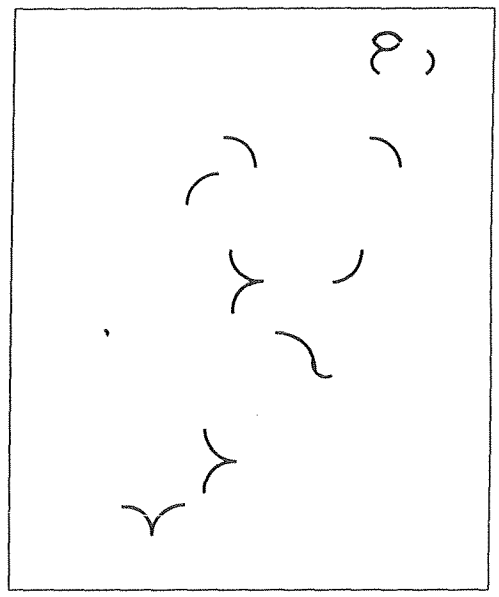
Nous présentons [FIG. 7.5] les résultats que nous obtenons en les comparant à ceux obtenus avec la méthode originale de Rosin et West. La figure [FIG. 7.5(b)] fait apparaître que de faux arcs peuvent être détectés avec la méthode de Rosin et West. Ces résultats sont imputables aux chaînes de petits segments fournis par la vectorisation, notamment au niveau des points de jonction. En utilisant notre méthode, ces arcs disparaissent, et certains cercles complets sont correctement détectés [FIG. 7.5(c)]. La position des arcs est également plus précise, même si cela n'est pas toujours flagrant sur la figure [FIG. 7.5(d)]. D'autres résultats sont présentés [FIG. 7.6, 7.7].

Des améliorations peuvent être apportées à notre méthode :

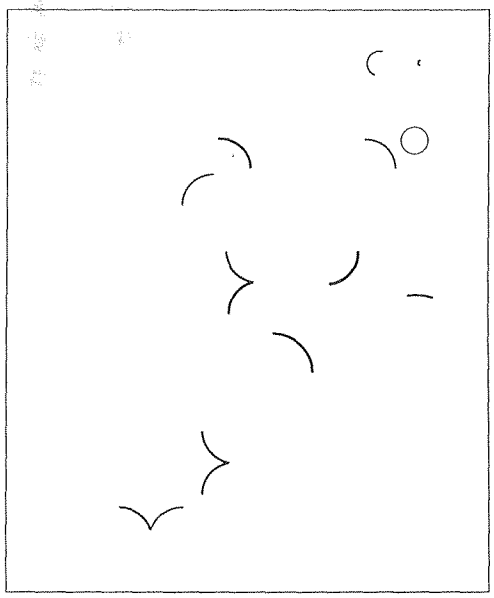
- Il faudrait générer et tester les hypothèses d'arcs sur plus d'une polyligne à la fois. Cela permettrait de détecter par exemple les arcs qui intersectent d'autres primitives graphiques, ou de reconnaître deux arcs dans le cas où ils partagent un petit segment, comme ceux pointés par la flèche de la figure [FIG. 7.5(d)]. La principale difficulté ne se situe pas au niveau de la méthode, qui demeure inchangée, mais réside dans le choix d'une structure de données permettant de maîtriser la complexité de l'implémentation.



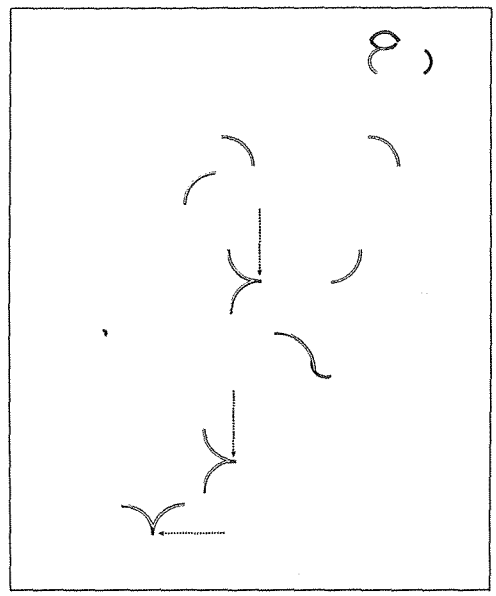
(a) Image originale.



(b) Détection d'arcs par la méthode de Rosin et West.

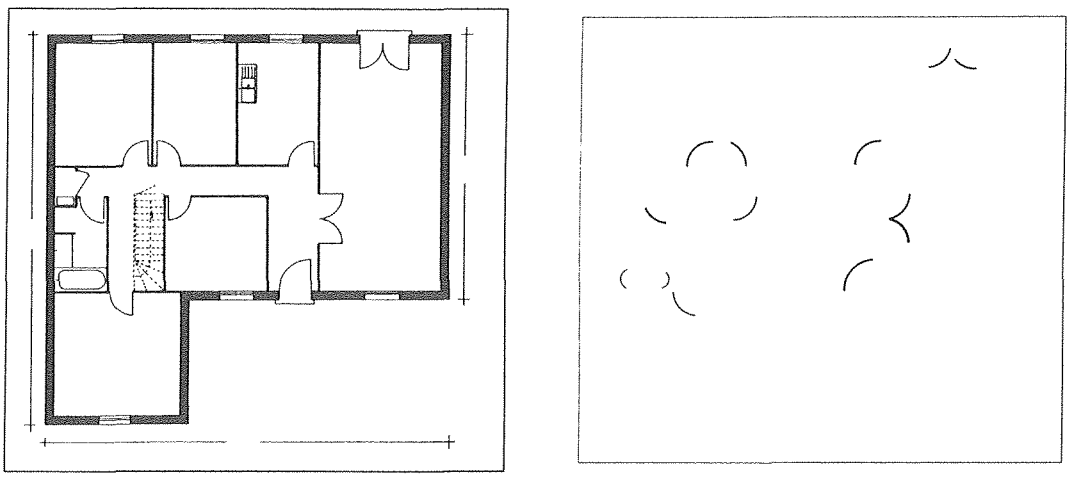


(c) Détection d'arcs par notre méthode.



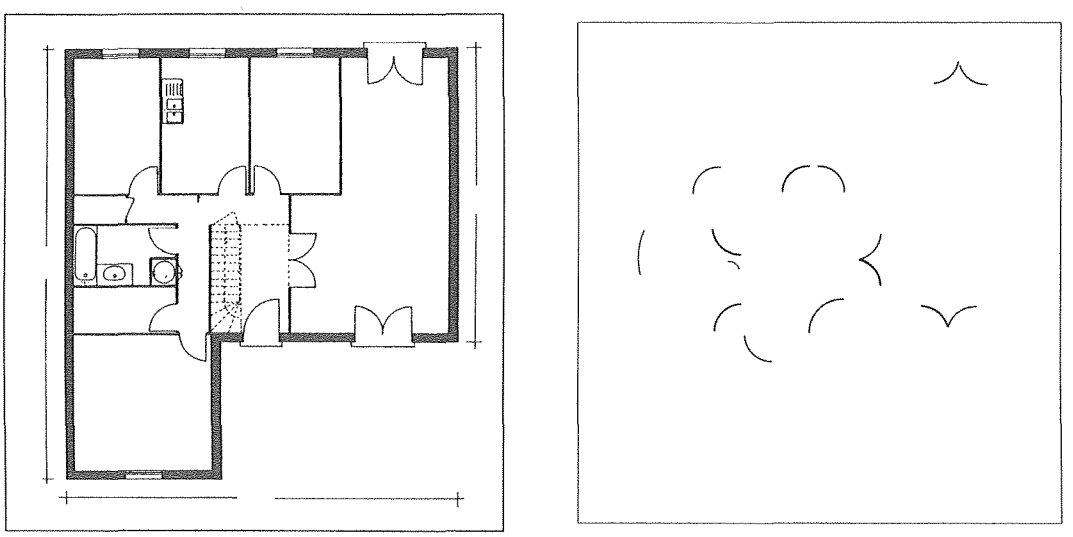
(d) Superposition des deux résultats.

FIG. 7.5 – *Résultats et comparaison de notre détection d'arcs avec celle de Rosin et West.*



(a) Image originale. (b) Arcs détectés.

FIG. 7.6 – *Résultats obtenus à partir des plans d'un pavillon.*



(a) Image originale. (b) Arcs détectés.

FIG. 7.7 – *Résultats obtenus à partir des plans d'un autre pavillon.*

- Une autre limitation de notre méthode réside dans sa dépendance d'un certain nombre de seuils, notamment celui utilisé pour la mesure de similarité entre deux mesures angulaires. Une amélioration envisageable serait d'étendre le travail de Rosin et West pour définir une mesure plus significative pour les arcs [Ros 97].

L'extraction d'arcs reste cependant un problème ouvert, dû à la complexité de l'équation du cercle par rapport à celle du segment. D'autre part, il subsiste des limitations, comme la détection des arcs de faible courbure par exemple, ou des arcs de petit périmètre pour lesquels il n'y pas assez d'informations pour débuter une hypothèse.

Quatrième partie

Reconstruction 3D

Chapitre 8

Détection de symboles : état de l'art

8.1 Introduction

Les symboles sont utilisés pour représenter de manière simple et compacte les éléments d'un type de document donné, ce qui leur confère une forte connotation sémantique. Ils se trouvent ainsi au cœur de la plupart des systèmes de reconnaissance graphique. À chaque type de document technique correspond un ensemble de symboles, plus ou moins normés, qui permettent de le décrire. Dans le cas des plans d'architecte, qui sont généralement des documents qui émanent du dessin technique et du dessin artistique, cette notion de norme est un peu floue. Il n'y a pas de consensus sur la représentation d'un symbole particulier, même si de grandes tendances se dégagent.

La détection de symboles consiste à rechercher les symboles propres à chaque catégorie de documents techniques. Les primitives graphiques résultant de la vectorisation, les segments et les arcs, constituent l'alphabet nécessaire à la représentation des symboles, quel que soit le type de document technique considéré. En considérant un plan architectural décrit à partir de telles primitives graphiques, nous souhaitons maintenant obtenir une description de ce même document à partir des symboles qui lui sont caractéristiques. L'objectif est de trouver le regroupement de primitives graphiques permettant de représenter les symboles que nous recherchons. Pour nos besoins, nous souhaitons décrire chaque plan en termes de murs (porteurs, cloisons), de menuiseries (portes, fenêtres), d'escaliers, de conduites. Ces symboles nous permettent de représenter la connaissance que nous recherchons sur les plans que nous traitons. Il existe d'autres symboles, tels que ceux liés à la plomberie (canalisation, lavabos, sanitaires, baignoires), à l'électricité (câblages) et à d'autres corps de métiers (revêtements de sol). Ces symboles ne sont cependant pas nécessaires à la modélisation que nous souhaitons faire des bâtiments et nous n'en effectuons ainsi pas de recherche particulière.

Suivant le type de document traité, certains symboles sont parfois directement représentés par les primitives issues de la vectorisation, tandis que certains d'entre eux correspondent à des assemblages

de ces mêmes primitives. Les règles d'assemblage utilisées, leur nombre et d'une façon générale la complexité globale des symboles recherchés, prédispose l'analyse à un type de détection particulier. Il faut généralement mettre en œuvre une technique particulière pour détecter un symbole, ou une famille de symboles. Ces techniques sont nombreuses, à l'instar de la variété de symboles existants, et regroupées à l'intérieur de grandes familles de détection, que nous nous proposons de passer en revue.

Dans [Chh 98a], Chhabra dresse un état de l'art des méthodes de reconnaissance de symboles les plus utilisées suivant les différents domaines où la détection de symboles est nécessaire. Il met en particulier l'accent sur le fait que beaucoup de solutions *ad hoc* existent, les méthodes de détection étant en général spécifiques à la nature des symboles recherchés. Ces méthodes spécifiques peuvent être regroupées sous deux grandes familles d'approches : les approches statistiques qui travaillent à partir de la représentation *bitmap* du symbole [§ 8.2], et les approches syntaxiques et structurelles qui sont, elles, basées sur sa représentation vectorielle [§ 8.3]. Les travaux sont cependant tellement nombreux sur la reconnaissance de symboles, et la reconnaissance de formes en général, que les états de l'art deviennent vite spécifiques à un domaine particulier, voire à un type de méthodes. D'autres méthodes à vocation générique et flexible ont récemment vu le jour [§ 9.3.1], et ont en particulier été utilisées dans notre équipe par C. Ah-Soon pour développer son système de détection de symboles [§ 9.4]. Enfin, nous terminons notre étude des différentes méthodes de détection de symboles par les algorithmes génétiques [§ 8.4].

8.2 Approches statistiques

Cette catégorie d'approches est basée sur l'étude de mesures statistiques extraites sur un échantillon, qui correspond à une zone d'une image. L'ensemble de ces mesures constitue un vecteur caractéristique $\vec{x} = (x_1, \ldots, x_n)$, où les x_i correspondent aux différentes mesures. Le vecteur caractéristique est ensuite comparé à des vecteurs modèles. L'objectif est de déterminer à quelle catégorie l'échantillon appartient, en comparant son vecteur caractéristique à ceux des symboles modèles. Les résultats obtenus à l'aide de cette catégorie de méthodes sont souvent fonction de la pertinence avec laquelle sont choisis les caractéristiques discriminatoires. Deux grandes familles de méthodes de comparaison existent : les méthodes de classification statistiques [§ 8.2.1] et les méthodes connexionnistes [§ 8.2.2].

8.2.1 Les méthodes de classification statistiques

Pour un état de l'art complet des principes de mise en œuvre dans les méthodes statistiques de reconnaissance de formes, le lecteur pourra se reporter à [Fuk 93] ou à [Kit 87]. Ces méthodes sont basées sur la construction de classifieurs capables de discriminer différents types de symboles recherchés. La construction d'un classifieur nécessite deux étapes. La première, appelée phase d'*apprentissage*, consiste à collecter des échantillons de données et de délimiter les différentes classes de données qui regroupent ces échantillons. La deuxième, appelée phase de *test*, consiste à introduire après normalisation les échantillons dont la classe d'appartenance est connue dans le classifieur que l'on construit. On utilise ainsi pour cette construction différentes techniques d'analyse de données : extraction, regroupement, tests statistiques — en particulier l'estimation de l'erreur de Bayes —, modélisation...

Ce type de méthodes est utilisé par Doermann *et al.* [Doe 96] pour reconnaître des logos. Ils décrivent une méthode qui leur permet de déterminer si une zone candidate contient ou non un des logos qui se trouvent dans la base des logos référencés. Cette méthode est basée sur l'utilisation d'invariants algébriques et différentiels qui leur permettent de décrire les formes des logos modèles et candidats en leur associant une *signature*, et d'effectuer ensuite des mises en correspondance à partir de ces signatures.

Les deux types d'invariants permettent de traiter les situations où les logos sont isolés et complets, et les situations où ils peuvent être recouverts ou incomplets.

Soffer et Samet [Sof 98] se sont également intéressés à la reconnaissance de logos en utilisant un vecteur de caractéristiques composé de descripteurs sur la circularité, l'excentricité, la rectangularité, des rapports d'aire... Après segmentation des zones candidates pouvant contenir un logo, le vecteur de caractéristiques correspondant est calculé et devient la signature du logo. L'utilisateur a le choix entre différentes méthodes d'identification, suivant les caractéristiques qu'il souhaite privilégier, basées sur des distances euclidiennes pondérées. Les auteurs utilisent ce même vecteur de caractéristiques pour l'identification de symboles sur des cartes géographiques [Sam 98], avec cette fois une identification effectuée à partir d'une adaptation de la méthode des *k plus proches voisins*.

Afin d'identifier des symboles électroniques, Lin *et al.* [Lin 85] placent les zones candidates sur des grilles, dont la finesse dépend de la précision souhaitée, et effectuent des calculs de rapports d'aires qui permettent de caractériser les symboles. Les vecteurs caractéristiques des symboles modèles et candidats sont ensuite comparés par distance euclidienne.

Ce type d'approches est également utilisé pour l'identification de symboles musicaux. C'est le cas de la méthode proposée par Armand [Arm 93] qui après segmentation des symboles musicaux, par suppression des lignes de portée, effectue une modélisation de ces symboles. Cette modélisation porte sur des données quantitatives (nombre de pixels, aire du rectangle englobant, rapport de ces valeurs, moments d'inertie...) L'identification est ensuite effectuée par méthode des *k plus proches voisins*. Ce type de documents est également largement étudié par le biais d'approches connexionnistes [§ 8.2.2].

8.2.2 Les méthodes connexionnistes

Ces méthodes, très répandues pour l'analyse de caractères, se trouve être adaptées à certains types de symboles, les caractères n'étant qu'une famille de symboles particuliers. Elles travaillent à partir des données *bitmap* correspondant aux zones candidates pouvant contenir un symbole qui font alors office de vecteur caractéristique. Ce genre de système est basé sur la modélisation informatique de neurones, dont le rôle est de fournir un résultat qui dépend des valeurs de ses différentes entrées [Ale 90]. Les neurones modélisés sont organisés en réseaux comportant plusieurs couches, ce qui permet de représenter les différents traitements que l'on peut effectuer en cascade sur un ensemble d'informations. La première de ces couches, appelée *couche d'entrée*, est destinée à recueillir les informations à analyser. Suivent ensuite un nombre variable de *couches cachées* dont les entrées sont constituées par les sorties des couches précédentes. Le réseau de neurones se termine finalement par une *couche de sortie* qui délivre le résultat de l'analyse faite par le réseau [FIG. 8.1].

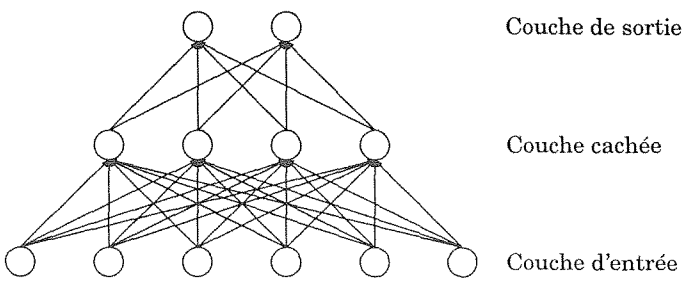


FIG. 8.1 – *Un exemple de modèle de réseau de neurones très utilisé : le perceptron multicouche.*

L'organisation même de certains types de réseau leur confère des propriétés intéressantes, dont la faculté d'agir comme des classifieurs après une période d'apprentissage sur les symboles à reconnaître. Pour un état de l'art des méthodes connexionnistes en reconnaissance de graphiques, le lecteur pourra se reporter à [Pao 93], qui présente les différentes architectures existantes, ainsi que les algorithmes correspondant.

Les réseaux peuvent alors être utilisés pour analyser différents types de symboles. C'est le cas des symboles présents sur les partitions musicales, au travers des travaux de Martin et Belissant [Mar 89, Mar 91], Miyao et Nakano [Miy 96], ou plus récemment de Anquetil *et al.* [Anq 99]. Dans tous les cas, la partition musicale est tout d'abord segmentée pour séparer les symboles musicaux des lignes de portée. Pour Martin et Belissant, les lignes de portée sont supprimées en utilisant une transformée de Hough, donnant par la suite des informations sur la localisation des symboles à rechercher, ce qui permet de définir des zones candidates à la détection de symboles. Celle-ci est ensuite effectuée à l'aide d'un réseau de neurones dont l'entrée est constituée par les caractéristiques des zones candidates. Pour Miyao et Nakano, la segmentation est réalisée en calculant un *histogramme des cordes*, c'est-à-dire l'histogramme résultant de la projection horizontale d'une portée, constituant l'entrée d'un réseau de neurones qui détermine les données à effacer pour isoler les symboles musicaux. Ceux-ci sont alors décomposés structurellement en éléments simples. Les notes sont directement obtenues par cette décomposition, tandis que les symboles restants sont classifiés par un second réseau de neurones. Pour Anquetil *et al.*, un réseau de neurones est utilisé après segmentation des portées non seulement pour la reconnaissance des symboles musicaux, mais également pour rejeter les symboles qui sont mal segmentés et qui peuvent ainsi être mal interprétés. Le système englobant cette analyse permet alors d'effectuer des retours en arrière et de resegmenter les zones candidates qui ont été rejetées.

Den Hartog et ten Kate [dH 94] utilisent les réseaux de neurones pour détecter les flèches se trouvant sur des documents généralement de mauvaise qualité. En effet, de par leur conception manuelle, ces flèches sont souvent irrégulières ou déformées, ce qui ne permet pas leur extraction par les techniques standards d'analyse de ces symboles. Un vecteur de caractéristiques est extrait pour chaque composante connexe candidate, basé sur les propriétés de la distance euclidienne du squelette de la composante. Ce vecteur est ensuite utilisé par le réseau de neurones pour l'extraction des flèches parmi les zones candidates.

Cesarini *et al.* [Ces 95] utilisent les réseaux de neurones pour la détection de symboles graphiques, tels que les logos. Les zones candidates sont localisées à partir de l'étude des composantes connexes et de l'utilisation de techniques de morphologie mathématique. Après normalisation, l'identification est effectuée à l'aide d'un ensemble de réseaux de neurones, un pour chaque symbole recherché. La matrice de points correspondant à la zone candidate est directement utilisée comme vecteur caractéristique.

La littérature la plus abondante relative à la reconnaissance de symboles à l'aide de réseau de neurones reste celle qui concerne la reconnaissance de caractères. Dans ce cadre, les réseaux de neurones sont maintenant souvent combinés avec d'autres types d'approches, comme les algorithmes génétiques pour définir des classifieurs [Lam 97] ou des architectures hiérarchiques regroupant plusieurs réseaux [Atu 99].

8.3 Approches syntaxiques et structurelles

Les approches statistiques sont basées sur la représentation des symboles modèles et candidats par des vecteurs de caractéristiques, construits à partir de mesures statistiques. De ce fait, l'information manipulée par ce type d'approches n'est pas très symbolique. Ce n'est pas le cas des approches syn-

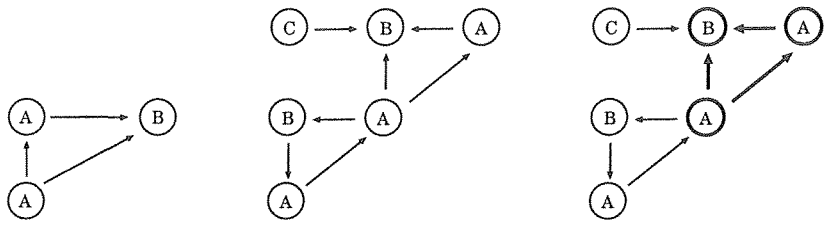
taxiques et structurelles, qui sont basées sur l'utilisation de structures de données abstraites (chaînes, arbres, graphes, listes) pour modéliser les données manipulées, ce qui leur confère un caractère plus symbolique [Bun 93a]. Ce type de représentation permet de définir des relations entre les différentes primitives. Ces relations peuvent être de type spatiales, conceptuelles, hiérarchiques, topologiques... Ce dernier type de relation permet de regrouper des primitives élémentaires en primitives plus évoluées, qui peuvent à leur tour être en relation avec d'autres primitives et constituer une partie de primitives encore plus évoluées. Ce processus mène à une certaine abstraction des données, manipulées sous une forme plus symbolique, ce qui facilite leur analyse et la compréhension globale des documents étudiés.

Dans ce type d'approches, la reconnaissance d'une forme ou d'un symbole est généralement obtenue par comparaison de sa représentation symbolique par rapport à celle des modèles prédéfinis. Les méthodes structurelles sont directement basées sur l'organisation hiérarchique des données, organisation traduite par des structures de graphes, et par les relations qu'elles entretiennent entre elles [Tom 96b]. Dans ce cas, la reconnaissance s'effectue par appariement des graphes modèles dans un graphe plus grand représentant l'ensemble des données à analyser. Cette technique, appelée *isomorphisme de sous-graphes* [§ 8.3.1], peut s'avérer très coûteuse en temps de calcul. D'autres méthodes sont basées sur les structures de graphes, comme la relaxation discrète [§ 8.3.2] ou encore la recherche de cliques maximales [§ 8.3.3]. Les méthodes syntaxiques reposent sur l'analogie de l'utilisation d'une structure arborescente entre la reconnaissance de formes et la syntaxe des langages [Fu 74]. Ce type d'approches permet de décrire des formes complexes à partir d'un petit ensemble de primitives et d'une grammaire décrivant les règles de composition de ces primitives [§ 8.3.4]. Les représentations à base de chaînes sont également largement utilisées [§ 8.3.5]. Enfin, nous exposons les limites liées aux approches syntaxiques et structurelles [§ 8.3.6].

8.3.1 Isomorphismes de sous-graphes

Les graphes permettent de représenter et de structurer différents éléments de façon hiérarchique, ainsi que de modéliser les relations qui existent entre ces différentes entités (connexion, position). Dans de tels graphes, les nœuds représentent les indices (point, contour, primitive) et les arcs les relations qui existent entre ces indices. Ces techniques sont largement utilisées, que ce soit en détection de symboles ou dans d'autres traitements de haut-niveau où des techniques de mises en correspondance sont requises (chapitre 12). Dans le cas de la détection de symboles, les modèles à rechercher et les données à analyser sont modélisés de cette manière, suivant les mêmes règles. Pour effectuer l'extraction de symboles dans le graphe représentant les données, il suffit alors de rechercher les occurrences des graphes modèles à l'intérieur du graphe représentant les données. Cette opération d'appariement est appelée *isomorphisme de sous-graphes*. Elle est implantée par une recherche arborescente et par une vérification de la cohérence avec les nœuds adjacents. Un des gros problème de cette approche est le coût de la recherche. En effet, ce type de problème est un problème NP-complet [Gar 79]. Nous détaillons ci-dessous différentes variantes existantes pour effectuer cette recherche.

- **Isomorphismes exacts** : cette technique effectue une recherche exacte d'un graphe modèle G_m à l'intérieur d'un graphe candidat G_c [FIG. 8.2]. La première méthode proposée pour cela est celle du *backtracking* [Ber 73]. Celle-ci consiste à tester de manière exhaustive tous les appariements possibles de G_m à l'intérieur de G_c en commençant par un nœud de G_m et en continuant avec les autres. Si l'appariement de G_m échoue lors de l'appariement d'un de ses nœuds, un retour en arrière dans la liste des appariements de nœuds est effectué pour explorer toutes les combinaisons possibles d'appariement, ce qui conduit aux tests de beaucoup de configurations inutiles.



(a) Modèle. (b) Candidat. (c) Appariement.

FIG. 8.2 – *Isomorphisme de sous-graphes exact. Le graphe modèle reconnu à l'intérieur du graphe candidat est représenté avec des traits forts.*

Afin d'éviter ces tests inutiles dans le parcours exhaustif des appariements possibles, Ullman propose l'algorithme de *forward checking* [Ull 76]. Dans cet algorithme, un test préalable sur les nœuds du graphe permet de réduire le nombre de candidats à examiner dans la suite de l'appariement et ainsi de réduire le temps de traitement nécessaire pour la recherche d'isomorphismes.

Lee [Lee 92] utilise cette technique pour la reconnaissance de symboles électriques.

- **Isomorphismes inexacts** : la nature même des documents techniques, qui sont généralement des documents manuels, fait qu'ils sont naturellement bruités, et que ce bruit s'ajoute généralement aux incertitudes des techniques d'analyse d'image. Dans ces conditions, il est souvent difficile d'envisager des isomorphismes exacts, deux entités de la même classe ayant rarement exactement la même description structurelle. Pour prendre ces contraintes en considération, la recherche d'isomorphisme peut s'effectuer de manière inexacte, c'est-à-dire en prenant compte d'une marge d'erreur, comme le proposent Shapiro et Haralick [Sha 81]. Ils introduisent des opérations d'ajout, de suppression et de remplacement sur les nœuds et les arcs des graphes, en leur associant un coût. La distance entre deux graphes est alors définie comme la somme minimale des coûts des opérations nécessaires à appliquer au premier graphe pour obtenir le second. Une adaptation de la méthode de *backtracking* est utilisée pour la recherche des isomorphismes, en classant les différents appariements élémentaires possibles selon un ordre croissant d'erreur, ce qui permet de considérer en premier lieu les appariements les plus intéressants. L'erreur cumulée des différents appariements nécessaires à un isomorphisme ne doit pas dépasser un seuil pour que l'isomorphisme soit considéré comme valide. D'autres algorithmes, comme l'algorithme A* [Nil 80] permettent d'élaguer au fur et à mesure les ensembles de solutions qui dépasse le seuil d'erreur autorisé au cours de la recherche, ce qui permet d'abandonner rapidement les voies stériles. Pour des exemples d'applications, voir les travaux de Granger [Gra 85] ou plus récemment de Lladós [Lla 97a] dans un système d'analyse de plans architecturaux.

Pour diminuer le temps de calcul, des recherches par heuristiques ont été proposées. Cependant, si ces approches permettent d'élaguer l'arbre et de réduire le coût de la recherche, elles peuvent également mener à l'obtention de solutions non optimales. Le lecteur pourra se reporter à [Sue 92] pour une illustration de ces techniques.

8.3.2 Les méthodes de relaxation

Basées sur la propagation de contraintes, les méthodes de relaxation peuvent être utilisées pour la recherche des isomorphismes de sous-graphes, en permettant de se centrer sur les aspects importants du graphe. La convergence vers l'identification d'un symbole n'est pas toujours assurée avec ce type de méthodes : la consistance globale n'est pas garantie, même si elle est localement juste. On les fait donc généralement précéder d'un prétraitement permettant d'élaguer l'ensemble des hypothèses. On distingue deux types de relaxation :

- **relaxation discrète** : le but de cette méthode est d'éliminer des hypothèses d'appariement localement incohérentes. Dans notre équipe en particulier, Habacha [Hab 93] a développé un système de reconnaissance de symboles électriques en se basant sur l'algorithme AC4 mis au point par Mohr et Hendersen [Moh 86][27]. Ce système permet notamment de prendre en compte les incertitudes liées aux problèmes de bruit et de définir un score pour chaque appariement retenu, l'appariement final retenu étant celui qui a le meilleur score.
- **relaxation continue** : intuitivement, cette méthode a pour objectif d'affiner les hypothèses plutôt que de les élaguer, le discrétisation n'étant pas toujours souhaitable. Le but est alors d'optimiser l'étiquetage des nœuds en maximisant une mesure de consistance. Faugeras et Berthod en particulier [Fau 81] proposent l'optimisation d'une fonction qui permet de réduire l'ambiguïté des étiquettes et d'augmenter la cohérence avec les étiquettes voisines.

8.3.3 Les cliques maximales

Le but de cette méthode est d'extraire le plus grand sous-ensemble de nœuds mutuellement interconnectés dans un graphe de compatibilité. Dans ce graphe, les nœuds représentent des hypothèses de mise en correspondance entre des indices modèles et des indices candidats, et les arcs les relations de consistance qui existent entre les nœuds. Dans ce contexte, une *clique* est un ensemble de nœuds du graphe de compatibilité mutuellement interconnectés entre eux, et la plus grande clique correspond donc au plus grand de ces sous-ensembles [FIG. 8.3]. Cette dernière est alors considérée comme le meilleur appa-

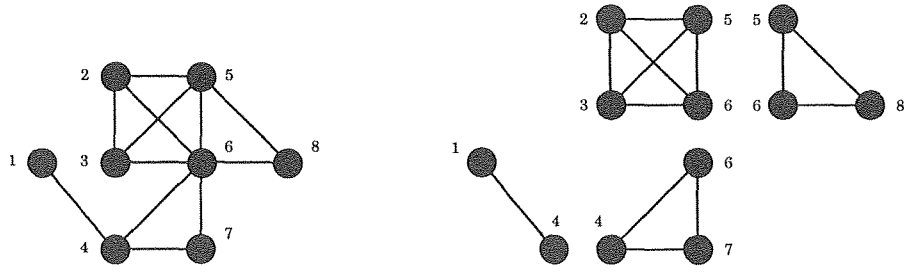


FIG. 8.3 – *Un exemple de graphe et des cliques qui en sont extraites. Sur cet exemple, la clique maximale est constituée des nœuds (2, 3, 5, 6).*

riement. L'avantage dans cette approche est qu'il n'est pas nécessaire de prendre en considération l'ensemble des caractéristiques associées à un symbole : on peut se limiter à un sous-ensemble suffisamment

27. Basé sur des contraintes binaires. Pour un algorithme basé sur des contraintes n-aires, voir l'algorithme GAC4 proposé par Mohr et Masini [Moh 88].

discriminant. La recherche de la clique maximale en elle-même est un problème NP-complet [Gar 79], et même si certains algorithmes plus ou moins sophistiqués existent pour optimiser cette recherche, on utilisera de préférence les cliques maximales dans des contextes où des graphes de compatibilité relativement petits sont générés.

Cette technique a été utilisée par Bolles et Cain dans leur méthode LFF[28] [Bol 83]. Dans cette méthode, ils se proposent d'identifier des pièces industrielles, qui peuvent se recouvrir les unes les autres. Les pièces sont caractérisées par des indices, représentés par les trous qu'elles comportent, par les coins, et par les relations locales de ces indices entre eux. La méthode est rapide car elle permet de discriminer les différents modèles de pièces non pas par une analyse globale, mais par une étude locale des indices et de leurs relations. Nous utilisons également cette approche pour mettre en correspondance les différents niveaux des bâtiments que nous étudions à l'issue de leur analyse 2D (chapitre 12).

8.3.4 Les grammaires

Les grammaires sont des méthodes syntaxiques, fondées sur la théorie des langages et des automates. Les grammaires formelles ont été initialement décrites par Chomsky [Cho 65], et adaptées à la reconnaissance de formes par Fu [Fu 74]. Les symboles terminaux de telles grammaires correspondent aux primitives issues de la vectorisation et les symboles non-terminaux décrivent les règles de production (de composition) des primitives. Les grammaires sont utilisées pour construire des analyseurs syntaxiques dont le but est d'assembler les primitives graphiques selon les règles de composition, afin d'identifier les symboles décrits. Elles présentent certains avantages, comme la possibilité de décrire des formes à structure récursive d'une manière compacte et aisée. Les grammaires formelles sont cependant limitées : elles n'acceptent pas les relations n-aires et sont mal adaptées, de par leur formalisme, à la représentation de données qui peuvent être bruitées.

D'autres types de grammaires, comme les grammaires attribuées [Knu 68], autorisent l'association d'informations sémantiques et d'attributs numériques aux règles. Les informations sémantiques sont utiles pour représenter les connaissances sémantiques qui existent au niveau des symboles, tandis que les attributs numériques permettent de prendre en compte les formes bruitées dans la modélisation des symboles. L'effet conjugué de ces informations permet d'améliorer les taux de reconnaissance de symboles lors de l'analyse.

Les grammaires présentées ci-dessus sont dites *linéaires* : elles permettent de représenter des relations unaires de concaténation entre les différentes entités (primitives ou composites). Ces relations, exprimées sous la forme *à gauche de* ou *à droite de*, ne permettent cependant pas de représenter les relations n-aires qui existent naturellement dans les formes bidimensionnelles. Pour permettre la représentation de telles relations, deux types de grammaire existent.

Le premier de ces types est illustré par le langage PDL[29] décrit par Shaw [Sha 69]. Celui-ci associe à chaque entité deux points caractéristiques, la *tête* et la *queue*, qui permettent de la manipuler. Tous les assemblages se font par l'intermédiaire de ces points caractéristiques, ce qui donne lieu à 4 possibilités, désignées sous forme d'opérateurs. Les grammaires linéaires correspondent, dans ce contexte, à une seule de ces possibilité (la relation *tête-queue*). Une extension de ce type de grammaire est utilisée par Masini et Mohr [Mas 83] pour l'analyse de dessins à main levée. Ils introduisent des opérateurs topologiques complétant le système de description, à partir duquel ils parviennent à analyser des dessins qui peuvent être bruités.

28. *Local Feature-Focus.*
29. *Picture Description Language.*

Les plex-grammaires permettent également d'utiliser des relations n-aires. Ces grammaires permettent de manipuler les formes par l'intermédiaire d'un nombre quelconque de points caractéristiques, et non plus à partir de deux seuls points [Fed 71]. Les règles de production stipulent alors les points caractéristiques qui sont conservés, et ceux qui sont superposés. Encore une fois, les grammaires linéaires ne sont qu'un sous-ensemble des plex-grammaires dans des conditions particulières. Ce type de grammaire est en particulier utilisé par Collin [Col 92] pour la détection de cotations dans des documents techniques.

D'autres types de grammaires existent encore et la littérature est abondante sur leur utilisation en reconnaissance de formes. Citons par exemple les web-grammaires, utilisées par Dori pour l'analyse de la cotation en dessin technique [Dor 88, Dor 89], les grammaires de graphes utilisées par Fahmy et Blostein [Fah 92] pour l'analyse de partitions musicales, Gonzalez et Wintz [Gon 87] qui illustrent l'utilisation de grammaires d'arbre par l'analyse de circuits électriques, ou encore Tsai et Fu [Tsa 80] qui utilisent des grammaires stochastiques pour modéliser plus facilement des formes bruitées.

8.3.5 Les chaînes

Cette technique se propose de représenter les symboles à modéliser par l'intermédiaire de leur contour. Ce contour est discrétisé en un ensemble ordonné de composants atomiques formant une *chaîne*. La reconnaissance des symboles modélisés est ensuite effectuée par une mise en correspondance de chaînes. Ce type de méthode permet de réduire la perturbation due au bruit et d'augmenter la rapidité de la recherche [Tsa 80]. L'algorithme de la mise en correspondance de chaînes est basée sur une optimisation par recherche récursive, issu d'un algorithme de recherche opérationnelle. Le lecteur pourra se reporter à [Wag 74] pour un document de référence pour la mise en correspondance de chaînes et à [Bun 92] pour un état de l'art complet des évolutions dans ce domaine.

Trois opérations élémentaires portant sur les symboles composant une chaîne permettent de passer d'une chaîne à l'autre : la *substitution*, l'*insertion* et l'*élision*, chacune associée à un coût. La *distance d'édition* entre deux chaînes est définie comme la séquence d'opérations de coût minimum à effectuer pour transformer la première chaîne en la seconde et conditionne l'identification de symboles candidats à partir d'un ensemble de symboles modèles. Un coefficient pondérateur est généralement utilisé en plus pour tenir compte de la longueur de la chaîne.

Un exemple d'application en reconnaissance de formes est donné par Wolfson [Wol 90]. Il présente deux algorithmes qui permettent de trouver la plus longue sous-partie de deux courbes qui sont à mettre en correspondance. Pour cela, une signature correspondant à une chaîne est extraite de chaque courbe en se basant sur les courbures, ce qui rend les signatures invariante à la rotation, à la translation et au changement d'échelle. Les signatures sont ensuite comparées en utilisant les techniques de mise en correspondance de chaînes. Tsai et Yu [Tsa 84] utilisent également la mise en correspondance de chaînes en définissant une opération supplémentaire, la *fusion*. Ils utilisent cette mise en correspondance pour la reconnaissance d'outils tels que des pinces.

8.3.6 Les limites

Sans connaissances *a priori* sur la structure des symboles à analyser, l'approche statistique est plus appropriée, bien que les différentes composantes du vecteur de caractéristiques ne sont pas toujours faciles à mettre en évidence, si cela est possible. Les méthodes syntaxiques et structurelles permettent, elles, de manipuler les symboles à rechercher comme des entités organisées. L'abstraction réalisée permet

de traiter plus facilement certains symboles qui sont alors décomposés en un ensemble de primitives et de relations entre ces primitives. Les caractéristiques sont donc plus faciles à mettre en évidence, et la méthode de reconnaissance dépend alors souvent de données telles que le nombre de symboles modèles à représenter... Toutefois, derrière cette apparente facilité de mise en œuvre pour certaines catégories de symboles, les approches structurelles et syntaxiques montrent leurs limites sur différents points :

- le temps de calcul : la majorité des recherches associées aux méthodes structurelles sont des problèmes NP-complets (pour les graphes) ou des problèmes P-complets (pour les arbres). Bien qu'il existe des algorithmes de recherches permettant d'élaguer l'espace des solutions, le coût de telles recherches est parfois dissuasif ;
- la prise en compte du bruit : les méthodes grammaticales en particulier se basent sur une description et une utilisation très rigoureuse des règles de composition des primitives. Il est donc délicat de faire intervenir le bruit dans de tels systèmes, surtout si la perturbation peut être forte. Le même problème se pose avec les isomorphismes de sous-graphes, quoique la recherche d'isomorphismes inexacts apporte partiellement une solution à ce problème ;
- les problèmes de segmentation : à partir du moment où des systèmes sont basés sur la manipulation et la composition de primitives graphiques, il importe que ces primitives soient bien détectées, en particulier à partir des phases de segmentation, mais également dans les phases d'extraction de primitives en elles-mêmes ;
- la représentation même des symboles : lorsque le modèle d'un symbole peut être formellement fixé, cela ne pose pas de problème de modélisation. En revanche, lorsque la modélisation d'un symbole dépend de la synthèse d'un grand nombre d'échantillons, il peut être difficile de représenter toutes les déclinaisons d'un symbole avec des approches syntaxiques et structurelles ;
- ces méthodes sont peu adaptées à la représentation de symboles difficiles à formaliser. L'exercice est d'autant plus délicat que les problèmes cités ci-dessus sont généralement accentués dans de telles conditions ;
- enfin, il apparaît finalement qu'en dépit de la méthodologie apportée par ce type d'approche, les systèmes créés sont finalement peu génériques. Les méthodes employées sont souvent *ad hoc* et les systèmes de reconnaissance peu transposables d'un problème à un autre. Certaines solutions ont cependant été apportées à ce problème [§ 9.3.1].

8.4 Algorithmes génétiques

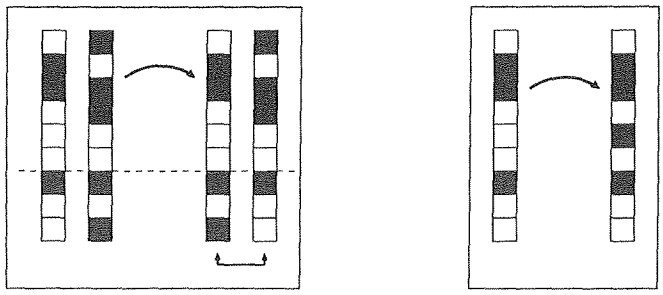
Les algorithmes génétiques sont inspirés de la génétique et de la théorie de la sélection naturelle. Ils ont pour but de modéliser des systèmes adaptatifs naturels et de construire des systèmes artificiels dotés des mêmes propriétés. Ces algorithmes sont exploités pour la robustesse de leurs recherches dans des espaces complexes [Gol 89]. Ils correspondent à une approche stochastique où le hasard des combinaisons est exploité afin d'obtenir un outil de recherche sophistiqué. Le principe général d'apprentissage des algorithmes génétiques se décompose en plusieurs phases, répétées jusqu'à l'obtention d'une population assez représentative de la connaissance à modéliser :

- Une population, un ensemble d'individus correspondant à des vecteurs de caractéristiques représentés sous forme de chaînes, est générée (semi-)aléatoirement[30]. Chaque individu de cette population représente une solution potentielle. À partir de cette population s'effectue la première phase,

30. Tout en restant aléatoire, la génération de la population initiale est généralement orientée afin de constituer une solution plausible.

dite de *sélection*, qui consiste à retenir un sous-ensemble de la population comportant des paires d'individus « intéressants », en fonction des caractéristiques étudiées.

- À partir de cette sélection, on effectue la deuxième phase dite de *reproduction* qui consiste à appliquer des opérateurs génétiques afin de créer de nouveaux individus. Cette phase a pour but de faire évoluer la population en propageant les caractéristiques des « meilleurs » individus, c'est-à-dire de ceux qui sont le plus adaptés à leur environnement. Les opérateurs génétiques les plus utilisés sont le *crossover* et la *mutation* [FIG. 8.4] ; ils sont associés à une probabilité d'application. La partie la plus délicate dans un algorithme génétique consiste à régler ces opérateurs : avec une trop faible probabilité, la faible évolution nuira au développement d'individus optimaux, tandis qu'avec une trop forte probabilité les individus générés seront trop perturbés, ce qui nuira à la capacité d'apprentissage.
- La phase d'*évaluation* calcule ensuite les caractéristiques de chaque individu créé.
- Vient enfin la phase de *remplacement* qui effectue le remplacement des plus mauvais individus de la population par les meilleurs. La population doit avoir une taille constante, ce qui induit une compétition entre les individus et permet de faire évoluer cette population.



(a) *Crossover.* (b) Mutation.

FIG. 8.4 – *Exemple d'application des opérateurs génétiques sur des vecteurs de caractéristiques, représentés par des chromosomes.*

Ce type d'algorithmes est utilisé avec efficacité pour la résolution de problèmes d'optimisations complexes. Ils ont cependant un coût d'exécution important sur les populations de grandes tailles.

Jiang *et al.* [Jia 99] utilisent des graphes pour modéliser des symboles graphiques et un algorithme génétique pour effectuer la reconnaissance de ces symboles. L'apprentissage se fait en calculant la médiane généralisée de l'ensemble des graphes, dont un des avantages est de lisser les imprécisions des différents échantillons. Cette opération est cependant très coûteuse et l'apprentissage est ainsi impossible avec des algorithmes classiques comme le A*. Les auteurs utilisent donc les algorithmes génétiques : ils représentent les graphes par l'intermédiaire de chromosomes, une fonction de corrélation et des opérations de *crossover* et de mutation. Ils obtiennent de très bons résultats en terme de rapidité avec cette méthode. Les algorithmes génétiques sont également utilisés conjointement avec d'autres approches de reconnaissance de formes. Citons Cross et Hancock [Cro 96], ainsi que Wang *et al.* [Wan 97], qui utilisent les algorithmes génétiques pour effectuer de l'isomorphisme de sous-graphes inexact. Les algorithmes génétiques autorisent dans ce cas l'implantation de stratégies de recherche locales qui permettent d'augmenter la vitesse de convergence vers une solution.

8.5 Approches hybrides

Les méthodes décrites dans ce chapitre ne sont pas toujours utilisées de façon « pure ». En fait, ces approches sont généralement combinées entre elles, chacune envisageant le problème de la reconnaissance de formes sous un angle différent. Cela peut mener à des schémas de type *prédiction-validation* où un premier type de méthodes est utilisé pour extraire un ensemble de candidats qui seront validés ou invalidés par une deuxième approche. C'est le cas par exemple de Bhattacharjee et Monogan [Bha 94] qui recherchent des symboles sur des cartes géographiques : une première étape statistique permet d'extraire des symboles candidats qui sont ensuite (in)validés par appariement structurel. Dans d'autres cas, les approches sont combinées pour améliorer leur efficacité. C'est typiquement le cas des recherches d'isomorphismes de (sous-)graphes, méthodes extrêmement coûteuses par des méthodes de recherche brute, qui peuvent être couplées avec des algorithmes génétiques, pour localiser plus rapidement un extremum local [Cro 96].

Dans ce contexte, il est peut-être inapproprié de classifier des méthodes de reconnaissance de formes comme hybrides, ce qualificatif pouvant potentiellement s'appliquer à la quasi-totalité des méthodes disponibles. Il nous a cependant paru intéressant de faire ressortir certaines méthodes dont l'hybridation constitue la base d'un système et non pas son amélioration ou son optimisation, tout en étant conscient que cette distinction est sujette à la subjectivité. C'est le cas de Tsai et Fu [Tsa 80] qui proposent d'utiliser des méthodes reposant sur des approches statistiques et structurelles pour traiter plus facilement des formes bruitées. Ici, l'hybridation est effectué dès le départ, au niveau même de la représentation des données à modéliser. Fu [Fu 83] et Tsai [Tsa 90] reprennent cette approche en la basant sur l'utilisation de grammaires attribuées.

8.6 Conclusion

Cet état de l'art fait apparaître l'existence d'une grande variété de méthodes pour la reconnaissance d'une grande variété de symboles. On peut noter qu'il n'existe pas de méthode générique permettant de s'adapter à la reconnaissance de n'importe quel symbole. Il est en effet souvent nécessaire de concevoir un système sur mesure, complètement ou partiellement, pour s'adapter au type de représentation des symboles, voire à d'autres facteurs, comme la qualité des données manipulées par exemple. La reconnaissance de symboles est donc un domaine très dynamique où beaucoup de problèmes restent ouverts. Si les approches présentées continuent à être régulièrement alimentées par de nouveaux travaux, la communauté de reconnaissance de symboles a également défini des objectifs en marge de toute approche :

- *Vers une évaluation de performances.* La diversité des symboles et des méthodes développées ne facilite pas la comparaison objective des performances de chaque méthode. Pour permettre une évaluation plus rigoureuse, des efforts sont entrepris pour créer des bases de données communes, regroupant des données représentatives sur lesquelles les méthodes peuvent être évaluées. Cet effort est nécessaire pour porter à maturité la reconnaissance de graphique, par la détermination de méthodes robustes.
- *Vers une plus grande généricité.* Les méthodes de reconnaissance *ad hoc* sont limitées : il est difficile de les transposer d'un système à un autre. Les chercheurs sont de plus en plus attentifs à l'élaboration de méthodes flexibles, permettant d'apporter une solution non seulement à un problème donné, mais également à la reconnaissance de symboles en général. Nous avons nous-mêmes entrepris des actions dans ce sens, qui ont mené à l'élaboration d'un système générique de reconnaissance de symboles, présenté chapitre 9.

– *Vers des interfaces homme-machine.* Les résultats présentés dans les travaux existants ne sont jamais parfaits et il semble utopique de pouvoir élaborer des méthodes présentant des taux de reconnaissance proches de 100 %. Il apparaît essentiel dans ce contexte de fournir des interfaces homme-machine convenables et de ne plus considérer la reconnaissance de symboles comme un processus automatique, mais assisté. La qualité des interfaces, et plus généralement des environnements produits, influe fortement sur la qualité des systèmes produits. Là encore, nous avons suivi ce principe pour la réalisation de notre système. Les aspects liés à l'interface homme-machine que nous avons développée, et plus généralement à la mise en œuvre de notre environnement d'analyse, sont présentés dans la partie 5 de ce manuscrit.

Chapitre 9

Reconnaissance de symboles architecturaux

9.1 Objectifs

Ce chapitre présente les techniques mises en œuvre pour détecter le type de symboles que nous recherchons : les symboles architecturaux. En effet, nous souhaitons pouvoir décrire chaque plan étudié en termes de murs (porteurs, cloisons), de menuiseries (portes, fenêtres), d'escaliers, de conduites. Il existe une grande variété de méthodes de détection, présentées chapitre 8. Nous détaillons ci-dessous les choix retenus pour certains symboles architecturaux.

Une des grandes difficulté de ce type de recherche réside en la représentation même des symboles. Même si les plans architecturaux sont des documents techniques, et héritent de la majeure partie des caractéristiques liées à ce type de documents, ils sont également des documents artistiques. Une des conséquences de cette affiliation est qu'il n'existe pas de standard de représentation des différents éléments architecturaux, l'architecte ayant toute liberté de représentation [FIG. 9.1]. Quelques tendances se dégagent tout de même pour la représentation d'un symbole particulier. Mais il n'est pas possible

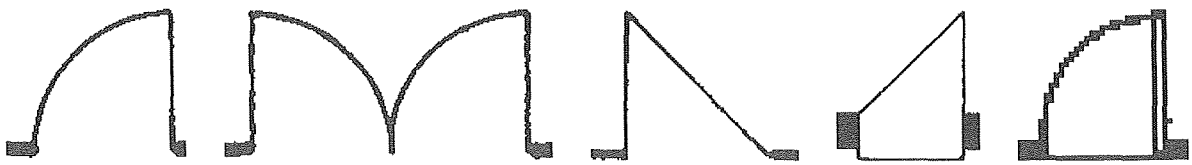


FIG. 9.1 – *Quelques exemples de représentation de portes.*

d'établir une bibliothèque exhaustive de symboles architecturaux pour autant. En effet, le symbole d'une porte sur un plan peut être associé à une fenêtre sur un autre, et les architectes utilisent parfois plusieurs symboles différents pour représenter un même élément architectural sur un plan, afin de différencier certaines déclinaisons de ce symbole (fenêtre simple ou double, matériaux de la porte...) En tout état de cause, il n'est pas possible de se baser sur un système où la description des symboles est codée de façon statique. La diversité de représentation, au niveau graphique et au niveau sémantique, nous pousse à nous orienter vers un système flexible de reconnaissance de symboles, dont la description doit être l'un des paramètres.

Nous nous intéressons dans un premier temps aux symboles dont la description est rudimentaire, comme c'est le cas par exemple pour les cloisons qui sont généralement simplement décrites par un simple segment, correspondant directement à l'une des primitives issues de la vectorisation [§ 9.2]. Nous présentons ensuite des méthodes de reconnaissance de formes qui ont été développées dans l'optique de la construction de systèmes plus génériques que celles exposées dans le précédent état de l'art [§ 9.3]. Ces méthodes, après adaptation à nos besoins, sont utilisées dans le système de reconnaissance de symboles développé dans notre équipe par Ah-Soon [§ 9.4]. Nous présentons finalement nos conclusions et perspectives dans ce domaine [§ 9.5].

9.2 Les symboles triviaux

Le symbole se différencie plus de la primitive graphique par sa valeur sémantique que par sa représentation. En effet, certains symboles sont représentés par une primitive graphique. En particulier, les murs porteurs sont représentés sur beaucoup de plans par de simples segments, généralement dotés d'une épaisseur supérieure de celle des autres segments, ce qui permet de les différencier. De même, lorsque les murs porteurs sont représentés par une zone hachurée, il n'y a pas réellement de détection de ce type de symboles : les primitives issues directement de la vectorisation, ou d'une des étapes suivant directement la vectorisation (détection de lignes pointillées, de zones hachurées) sont directement assimilables aux symboles recherchés ; dans ce cas, la détection est immédiate. Ainsi, lorsque les murs porteurs sont représentés par des traits forts, ils sont directement accessibles à partir de l'image de traits forts obtenue à partir de l'image originale [§ 3.3].

D'autres symboles, comme les cloisons sont obtenus par différence. En effet, ces symboles sont généralement représentés par de simples segments, attribués par une épaisseur qui ne permet pas de les différencier des segments entrant dans la composition d'autres symboles architecturaux, comme les menuiseries [§ 9.4]. Dans ce genre de situation, les recherches « élaborées » de symboles sont effectuées dans un premier temps. L'ensemble des primitives restantes, dans ce cas précis les segments, est considéré comme l'ensemble des derniers symboles recherchés. Nous revenons sur ce point lors de la fusion des symboles extraits d'un plan pour construire le modèle 3D correspondant (chapitre 11).

9.3 Des méthodes adaptées à nos besoins

9.3.1 Description générique des symboles

Comme nous l'avons évoqué [§ 8.3.6], il existe peu de méthodes génériques en reconnaissance de formes. Le type d'approche, statistique ou structurelle, ainsi que la méthode employée dépendent beaucoup des données à analyser, et il est souvent délicat de transposer un système d'analyse conçu pour un

type de symbole à un autre type de symbole. Ce point est particulièrement important, surtout dans notre contexte. En effet, il n'existe pas de standard de représentation de symboles sur les plans d'architecte et il est inconcevable de construire un nouveau système à chaque nouvelle représentation de symbole rencontrée.

C'est en partant du constat que beaucoup de solutions *ad hoc* sont proposées en reconnaissance de formes que Pasternak [Pas 94, Pas 96] s'est intéressé à la construction d'un système générique, le système ADIK. Le but de ce système est de fournir un noyau générique d'interprétation de symboles qui peut ensuite être adapté à un domaine d'application en donnant les descriptions des symboles à rechercher à partir de primitives graphiques. Les descriptions ne constituent alors plus qu'un paramètre du système au lieu d'en être le cœur. Ces descriptions sont formalisées au moyen d'un langage déclaratif permettant de définir :

- des contraintes sur des *attributs* : la position d'un point, la position, l'orientation ou la longueur d'un segment, d'un arc ou d'un texte,
- des contraintes sur la *composition* au moyen de relations géométriques entres les primitives : distance, proximité, connexité, angle.

Le langage de description suit de plus une philosophie orientée objet, autorisant la spécialisation de types déjà définis, ce qui permet de définir une taxinomie de symboles à rechercher. L'intérêt est double :

- la description des symboles est plus aisée car plus structurée, plus abstraite et plus compacte ;
- la recherche des symboles à partir des descriptions est ensuite facilitée et accélérée en tenant compte de ces relations de composition. En particulier, cela permet de factoriser la recherche d'une « famille » de symboles : une configuration de primitives étant définie à la fois comme un symbole et comme composante d'un symbole plus complexe ne sera testée qu'une seule fois lors de la recherche des symboles.

Le système proposé par Pasternak reste cependant limité, et les relations de factorisation ne sont pas exploitées au mieux, notamment parce que les relations de composition sont fournies et non pas calculées, ce qui ne permet pas de garantir une solution optimale.

9.3.2 Recherche factorisée d'isomorphismes de sous-graphes

Dans la famille des algorithmes permettant d'effectuer une recherche d'un sous-graphe modèle à l'intérieur d'un graphe candidat, beaucoup de méthodes ont été proposées [§ 8.3.1]. Toutes ces méthodes ont en commun d'effectuer la recherche d'*un* graphe modèle particulier à l'intérieur du graphe candidat. Différentes techniques ont été proposées afin de ne pas faire une recherche brute, c'est-à-dire exhaustive des différentes possibilités, mais d'optimiser cette recherche en élaguant les voies stériles dès qu'elles sont détectables. Il y a cependant peu de travaux qui ont été effectués pour optimiser la recherche globale de différents graphes modèles à l'intérieur d'un graphe candidat. En effet, lors de la recherche d'un isomorphisme, la complexité de la recherche est proportionnelle à la taille du graphe candidat, mais également à la taille et au nombre de graphes modèles à tester, c'est-à-dire au nombre de symboles modèles existants.

Pour cela, Messmer et Bunke [Mes 93, Bun 95, Mes 96] proposent de représenter tous les graphes modèles sous une forme compacte, autorisant leur recherche conjointe à l'intérieur d'un graphe candidat. Leur système combine des techniques de reconnaissance de formes avec des concepts d'apprentissage afin de résoudre le problème de la reconnaissance de symboles dans des documents techniques. Les symboles, ainsi que le document étudié, sont représentés par des graphes attribués et la mise en cor-

respondance est effectuée par un isomorphisme de sous-graphes inexact. Le principe de la méthode est de décomposer les graphes modèles en sous-graphes de plus en plus élémentaires jusqu'à l'obtention d'une décomposition en graphes d'un seul nœud. Lors de la décomposition de chacun des modèles, il est possible de trouver certains sous-graphes communs à plusieurs graphes modèles. L'idée est de mettre en facteur ces sous-graphes dans la reconstruction d'un graphe contenant la description de tous les graphes modèles utilisés.

Pour cela, toutes les représentations sont regroupées au sein d'un seul graphe, organisé comme un réseau. Dans ce réseau, on dénombre trois types de nœuds :

- Les nœuds *racines* qui constituent les points d'entrée dans le réseau. Ils représentent des sous-graphes élémentaires constitués d'un seul nœud, un pour chaque type d'étiquettes[31]. Ces nœuds transmettent l'ensemble des primitives élémentaires correspondant à la description d'un document à leurs fils, qui sont d'un des types énumérés ci-dessous.
- Les nœuds *testeurs* qui permettent de tester une contrainte relative aux sous-graphes correspondant à leurs deux nœuds pères. Cette contrainte est la traduction d'une règle de composition, ce qui permet d'étendre les deux sous-graphes auxquels ils sont reliés. Ce regroupement produit un nouveau sous-graphe qui résulte de la fusion des sous-graphes pères et du nœud courant. Ce nouveau sous-graphe est ensuite utilisé dans la composition d'autre sous-graphes.
- Les nœuds *terminaux* qui ne sont reliés qu'à un seul père et à aucun fils. Ces nœuds correspondent au symbole modèle décrit par le sous-graphe auquel ils sont reliés.

Ce type de réseau est construit de façon incrémentale en ajoutant une à une les descriptions correspondant aux modèles utilisés. À chaque nouveau modèle, ne sont ajoutés au réseau que les arcs et les nœuds *nécessaires* à la description du modèle et *absents* du réseau jusqu'à présent. Une fois ce réseau créé, la détection de modèles est amorcée en introduisant l'ensemble des primitives par l'intermédiaire des nœuds racines. Ces primitives traversent alors le réseau en passant par les nœuds dont elles vérifient les contraintes, et en se regroupant au fur et à mesure en sous-graphes représentatifs des contraintes respectées. Les sous-graphes ainsi constitués qui atteignent les nœuds terminaux du réseau correspondent aux modèles détectés. Ce type de réseau est particulièrement performant sur les « familles » de modèles constitués de bases communes. L'ajout de nouveaux modèles n'entraîne dans ce cas que de légères modifications du réseau. La compacité et les performances de recherche de ce réseau dépendent de sa capacité de factorisation des modèles à représenter.

9.4 Reconnaissance de menuiseries

Nous décrivons dans ce paragraphe les travaux réalisés par Ah-Soon pour effectuer la détection de symboles à partir de plans d'architectes vectorisés. Ce processus, qui est un maillon indispensable dans l'analyse de plans architecturaux, est décrit dans ses grandes lignes et quelques résultats sont présentés. Le lecteur pourra se rapporter à [AS 98b] pour des explications plus détaillées et des exemples de construction et d'utilisation de ce système.

31. Étiquettes qui correspondent aux types de données stockées dans les nœuds du réseau. Dans notre contexte, ces étiquettes correspondent aux différentes primitives graphiques décrivant un document, c'est-à-dire aux segments et aux arcs.

9.4.1 Description des symboles

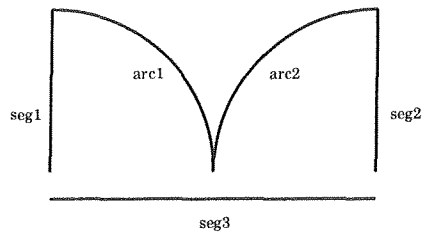
Au vu de la nature peu normée des symboles architecturaux, le système de description des symboles proposé par Pasternak [§ 9.3.1] semble particulièrement adapté à nos besoins. En particulier, la description des symboles, basée sur le langage ADIK, est facilitée car elle permet d'exprimer cette description par décomposition des symboles en primitives graphiques. Cette opération, qui est relativement instinctive pour un utilisateur — en tout cas sur des symboles simples, ce qui est notre cas —, permet d'adapter rapidement un système de reconnaissance de symboles à un type de plan donné à traiter. Le système que nous utilisons s'est donc tout naturellement inspiré de ce type d'approche, qui permet une souplesse d'utilisation nécessaire à notre objectif. Afin de modéliser les symboles que nous recherchons, un nouveau langage de description a été introduit. Celui-ci à été conçu à partir des hypothèse suivantes :

- les primitives graphiques dont nous disposons sont de deux natures : des segments et des arcs. Ces primitives, par composition, permettent de décrire les symboles auxquels nous nous intéressons. La composition peut être basée sur des relations de connexité qui existent entre plusieurs primitives ; celles-ci sont donc dotées d'attributs permettant d'accéder à leurs extrémités ;
- les primitives peuvent être regroupées en suivant des contraintes :
  - portant sur une primitive unique, permettant de « filtrer » une primitive sur des critères tels que la longueur, l'épaisseur, l'angle d'ouverture (pour les arcs) ;
  - impliquant deux primitives, dans le but de les composer. Ces contraintes portent sur la connexité entre ces primitives, la comparaison de leurs attributs ou leur distance relative ;
- les symboles que nous recherchons sur les plans d'architectes sont *a priori* les suivants : des portes (simples ou doubles) et des fenêtres (simples ou doubles). Le système doit cependant pouvoir s'adapter aisément à tout autre symbole architectural à détecter ;
- les symboles sont composés d'un nombre fini de primitives dont le type est précisé.

Ces hypothèses ont servi de base à la définition d'un langage, dont la grammaire est présentée ci-dessous. Il est possible d'établir ensuite la description des symboles à rechercher à l'aide de cette grammaire. Les descriptions sont stockées dans des *fichiers texte*, ce qui permet de les définir et de les réutiliser aisément.

symbole	$\mapsto$	**# type** SEG *entier* ARC *entier* {**contrainte**;}$^+$ **instanciation**
contrainte	$\mapsto$	**contrainte** ou **contrainte** | **contrainte** et **contrainte**
contrainte	$\mapsto$	**point** == **point** | **réel op_comp réel**
réel	$\mapsto$	*réel* | *réel* cm | min(**réel** , **réel**) | max(**réel** , **réel**)
réel	$\mapsto$	abs(**réel**) | sin(**réel**) | cos(**réel**) | tan(**réel**)
réel	$\mapsto$	**point**.x() | **point**.y() | **point**.distance(**point**)
réel	$\mapsto$	**prim**.f_elem() | **prim**.f_angle(**prim**) | **réel op_alg réel**
réel	$\mapsto$	**arc**.rayon() | **point**.angle(**point**, **point**)
f_elem	$\mapsto$	longueur | épaisseur | x | y
f_angle	$\mapsto$	pt1_angle_pt1 | pt1_angle_pt2 | pt2_angle_pt1 | pt2_angle_pt2
point	$\mapsto$	**prim**.point1() | **prim**.point2() | **arc**.centre()
prim	$\mapsto$	**arc** | **segment**
arc	$\mapsto$	arc *entier*
segment	$\mapsto$	seg *entier*
op_comp	$\mapsto$	$< \mid > \mid <= \mid >= \mid =$
op_alg	$\mapsto$	$* \mid / \mid + \mid -$
instanciation	$\mapsto$	**type** ({ **réel** | **point** } { ,**réel** | ,**point** }*)
type	$\mapsto$	porte | porte_double | fenêtre | fenêtre_double | *chaîne*

Une description est composée de trois grandes parties : un en-tête indiquant le nombre et la nature des primitives composant le symbole, une liste de contraintes qui décrivent les relations existant entre les primitives et une partie qui indique comment instancier le symbole associé à cette description. Un exemple de symbole et de sa description associée sont présentés [FIG. 9.2].




```
#porte_double SEG 3 ARC 2
seg1.pt1_angle_pt1(arc1) = 90;
arc2.pt1_angle_pt1(seg2) = 90;
arc1.pt2_angle_pt2(arc2) = 0;
arc1.angle() = 90;
arc2.angle() = 90;
30 cm < seg1.longueur();
30 cm < seg2.longueur();
arc1.centre() = seg1.point2();
arc2.centre() = seg2.point2();
seg3.point1().distance(seg1.point2()) = 30;
seg3.point2().distance (seg2.point2()) = 30;
porte_double(arc1.centre(), arc2.centre())
```

FIG. 9.2 – *Un symbole et sa description associée.*

9.4.2 Définition d'un réseau de contraintes

Principe

Le réseau proposé par Messmer et Bunke [§ 9.3.2] a été conçu pour effectuer des recherches optimisées d'isomorphisme de sous-graphes, notamment en factorisant autant que possible les descriptions des graphes modèles. Le système d'analyse de symboles que nous décrivons ici n'est pas basé sur une représentation des symboles modèles par des graphes, mais sur l'utilisation d'une grammaire exprimant des contraintes. Il a semblé cependant intéressant de s'inspirer du réseau proposé par Messmer et Bunke, en particulier pour la factorisation de la représentation des modèles qui peut être utilisée pour factoriser les contraintes communes à un ensemble de symboles [AS 97, AS 98a, AS 98d].

Adapté à notre contexte, un tel réseau est utilisé pour décrire les contraintes utilisées pour détecter les symboles que nous recherchons. De ce fait, le fonctionnement est le suivant : les primitives graphiques correspondant à la description d'un document sont introduites dans le réseau et se propagent à travers les nœuds dont elles vérifient les contraintes. On appelle *indice* un ensemble de primitives respectant un ensemble de contraintes. Initialement, ces indices correspondent aux primitives introduites dans le réseau. Les indices qui traversent le réseau jusqu'à un de ses nœuds terminaux correspondent aux symboles recherchés. Le réseau est composé de nœuds dont le type est l'un de ceux enumérés ci-dessous :

- Les nœuds *racines*. Il en existe un pour chaque type de primitives graphiques que nous manipulons. Il existe ainsi un nœud appelé `NRSegment` et un nœud appelé `NRArc`. Ces nœuds sont utilisés pour introduire les primitives graphiques décrivant le document à anlyser dans le réseau. Ils ne possèdent pas de père et ont autant de fils qu'il y a de contraintes à vérifier directement sur les primitives graphiques.
- Les nœuds *testeurs*. Ces nœuds expriment les contraintes qui décrivent les symboles modélisés. Il en existe de deux sortes différentes :
  - Les nœuds `NRCondition` : ces nœuds possèdent un seul père. Ils sont chargés de tester une contrainte, exprimée sous la forme d'un prédicat, sur les indices qu'ils reçoivent de leur père. Ils transmettent ensuite les indices qui respectent cette contrainte à leur nœuds fils.

- Les nœuds `NRFusion` : ces nœuds possèdent deux pères. Ces nœuds, comme les nœuds `NRCondition`, sont chargés de vérifier une contrainte qui s'applique cette fois sur deux indices. Ils ont donc un rôle de composition. Tous les indices reçus du premier père sont fusionnés deux à deux avec tous les indices du second père à la condition :
  1. qu'ils vérifient la contrainte exprimée par le nœud ;
  2. que les deux indices comportent des ensembles disjoints de primitives, une même primitive ne pouvant être utilisée deux fois dans la composition d'un symbole.
- Les nœuds *terminaux*. Ces nœuds, appelés *NRTerminal*, correspondent aux différents symboles modèles existants. Ils possèdent un père et n'ont pas de fils. Les indices qui parviennent jusqu'à l'un de ces nœuds correspondent au symbole décrit par le nœud considéré.

Réalisation

Le processus de recherche, correspondant à la propagation des primitives et ensuite des indices à travers le réseau, a un coût qui est fonction des éléments suivants :

- *La taille du réseau.* Cette taille dépend elle-même du nombre de symboles et de la diversité de représentation de ces symboles. Ainsi, une famille de symboles présentant des caractéristiques de représentation communes engendrera un réseau plus compact qu'une famille de symboles dont la représentation est hétérogène. Dans les deux cas, la taille du réseau croit en fonction du nombre de symboles.
- *Le nombre de primitives.* Plus ce nombre est élevé, plus il y aura d'hypothèses de symboles générées. De plus, les nœuds `NRFusion` combinant les primitives entre elles, on obtient rapidemment une explosion combinatoire, même sur une image de taille moyenne.

Afin de maîtriser cette explosion combinatoire, la recherche de symboles n'est pas effectuée globalement, mais localement sur l'image. Cette focalisation n'est pas gênante car la taille des symboles est relativement petite par rapport à la taille de l'image et les différentes primitives qui composent un symbole sont proches les unes des autres. La recherche ne s'effectue donc pas sur l'image, mais à l'intérieur d'une fenêtre de recherche, qui est déplacée au fur et à mesure du traitement de l'image afin de parcourir entièrement cette image.

Le processus de recherche des symboles est également affecté par le bruit inhérent aux primitives délivrées par la vectorisation. Dans ces conditions, il est quasi-impossible de détecter des symboles qui vérifient rigoureusement les contraintes exprimées sur leur description. La recherche doit donc être effectuée de manière inexacte, en tenant compte d'une tolérance dans la vérification des contraintes. Ainsi, une mesure d'erreur est effectuée à chaque vérification de contraintes, mesure qui se propage avec l'indice à l'intérieur du réseau. Lorsque la valeur de l'erreur atteint un certain seuil, fixé expérimentalement, l'indice correspondant n'est plus considéré dans le processus de détection de symboles. Des règles de calcul et de composition d'erreur en fonction de la nature des contraintes sont définies. Le lecteur pourra se reporter à [AS 98b] pour plus de détails sur la définition et la gestion de ces erreurs.

9.4.3 Construction du réseau

La construction du réseau, largement inspirée du réseau proposé par Messmer et Bunke, a pour but de créer un système de vérification de contraintes menant à la détection de symboles à partir d'un ensemble de primitives graphiques. Les qualités attendues de ce réseau sont la compacité, c'est-à-dire la capacité

de factorisation des descriptions de symbole, et la généricité, c'est-à-dire l'adaptation aisée à un jeu de symboles donnés. Pour ce dernier point, une solution a déjà été proposée grâce au langage de description introduit [§ 9.4.1]. Pour la compacité, il est nécessaire d'introduire certaines heuristiques.

À partir d'un ensemble de symboles à reconnaître, il est possible de construire plusieurs réseaux différents, chacun correspondant à une décomposition et à une factorisation différentes de l'ensemble des graphes modèles. Ces différents réseaux ne présentent cependant pas les mêmes performances. Pour créer un réseau assez compact, Messmer propose une approche itérative et incrémentale [Mes 93]. À chaque ajout de la description d'un symbole à reconnaître dans le réseau, une recherche permet de déterminer quelles sont les composantes de ce symbole (des sous-graphes) que le réseau est déjà en mesure de reconnaître. L'ajout du symbole dans le réseau s'effectue alors par les actions suivantes :

- ajout de nœuds exprimant les contraintes du symboles qui ne sont pas déjà représentées dans le réseau,
- ajout de nœuds permettant de raccorder des nœuds existant, correspondant à des contraintes,
- ajout d'un nœud terminal correspondant au symbole à reconnaître.

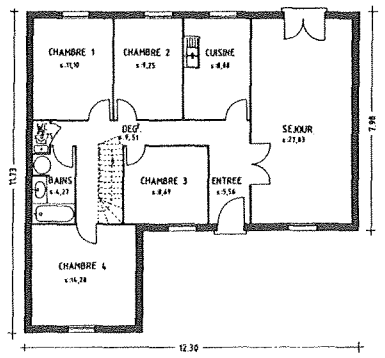
Certaines heuristiques ont été ajoutées à ces principes pour la création du réseau de contraintes que nous utilisons. En particulier, les contraintes simples (correspondant aux nœuds `NRCondition`) sont ajoutées en tête de réseau. Ainsi, les contraintes se reportant à une seule primitive graphique sont testées en premier, ce qui permet de mieux maîtriser l'explosion combinatoire du nombre d'indices créés. En effet, cela diminue le nombre d'indices à fusionner dans la suite du réseau, au niveau des nœuds `NRFusion`.

9.4.4 Utilisation du réseau — Résultats

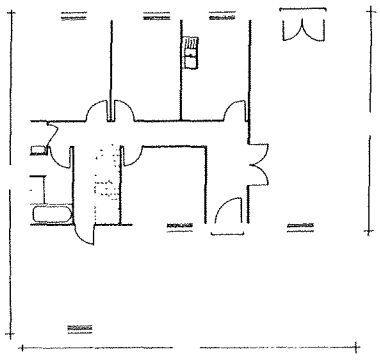
Une fois un réseau créé à partir d'un fichier de description de symboles, il est possible d'utiliser ce réseau pour reconnaître les symboles d'un plan donné. Pour cela, il est nécessaire de fournir, outre l'image vectorielle correspondant au plan, les dimensions de la fenêtre de recherche et un seuil correspondant à la marge d'erreur tolérée. Quelques résultats obtenus par l'utilisation d'un réseau de contraintes sont présentés [FIG. 9.3]. Les résultats obtenus sont assez satisfaisants, notamment sur les deux premiers plans présentés où le taux de reconnaissance est supérieur à 90 %. Les résultats présentés sur le troisième plan sont un peu plus bruités, ce qui est dû en partie à la simplicité de description des symboles. En effet, ceux-ci sont généralement composés d'un nombre limité de primitives graphiques, disposées selon les directions principales (les axes ainsi que les diagonales). D'autres symboles — dans ce troisième exemple, les marches d'escalier — peuvent alors partager certaines de ces caractéristiques et être reconnues lors de l'analyse. Il serait nécessaire d'injecter d'autres connaissances pour discriminer ces symboles. Par ailleurs, si l'analyse reconnaît plus de symboles que souhaitable dans certains cas, elle peut également ne pas reconnaître des symboles recherchés. Dans ce cas, l'explication est essentiellement imputable à la qualité des résultats issus des étapes précédentes, notamment de la vectorisation.

9.5 Perspectives

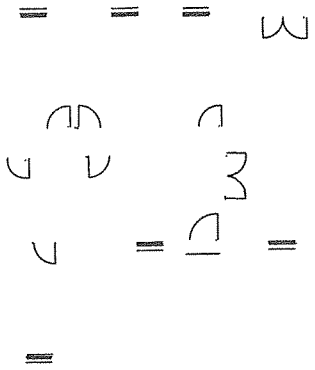
Suite à ces travaux, nous souhaitons améliorer ce système de reconnaissance de symboles, afin de pouvoir permettre la reconnaissance d'un plus grand nombre de symboles. En effet, si le système permet d'obtenir de bons résultats sur un jeu réduit de symboles à rechercher, ses performances chutent lorsque le nombre et la complexité des symboles augmente. À ce titre, il serait intéressant d'effectuer une évalua-



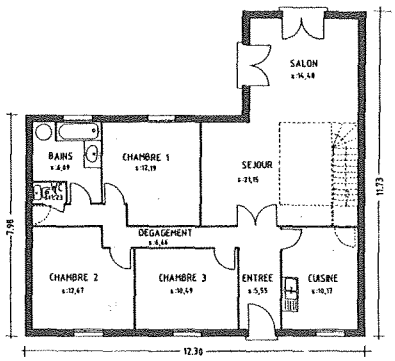
(a) Un plan.



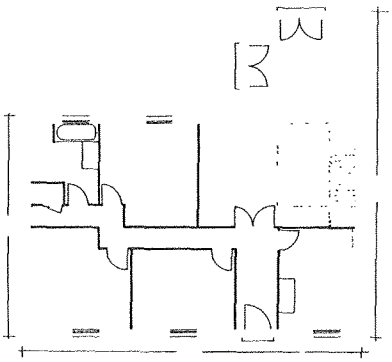
(b) Ses traits fins.



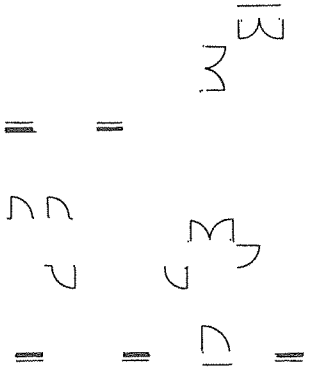
(c) Ses symboles.



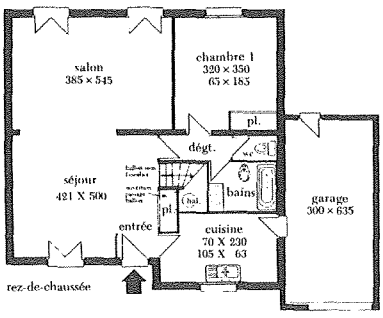
(d) Un plan.



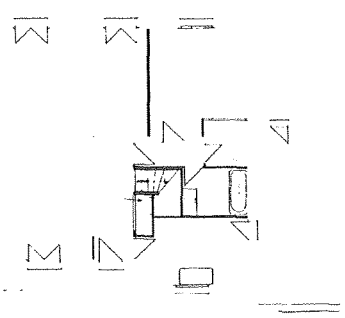
(e) Ses traits fins.



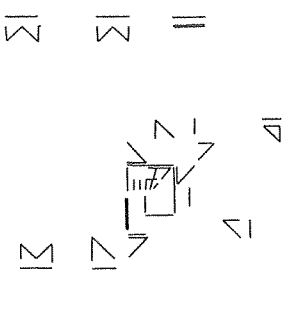
(f) Ses symboles.



(g) Un plan.



(h) Ses traits fins.



(i) Ses symboles.

FIG. 9.3 – *Résultats de la recherche de symboles.*

tion de performances plus complète afin de déterminer précisemment les limites de ce type d'approche. La reconnaissance de symboles peut également être approchée par d'autres techniques, dont certaines ont été présentées chapitre 8. Parmi elles, nous envisageons d'explorer d'autres méthodes pour cette étape délicate, comme les algorithmes génétiques. Cette technique, déjà utilisée en reconnaissance de symboles, permet d'obtenir d'assez bons taux de reconnaissance sur des données bruitées, problème auquel nous sommes confrontés. Le lecteur pourra en particulier se reporter aux travaux de Jiang *et al.* [Jia 99] pour une illustration de l'utilisation d'algorithmes génétiques pour réaliser de l'isomorphisme de sous-graphes.

Chapitre 10

Recherche des cages d'escaliers

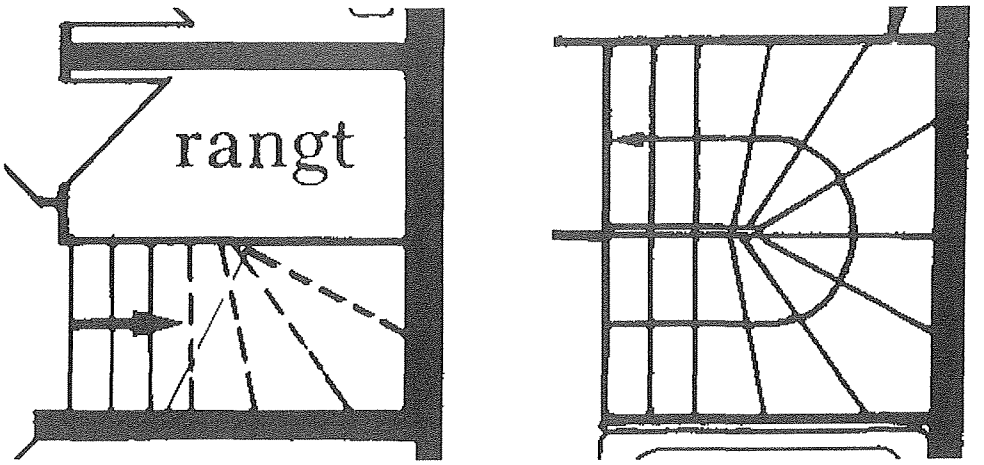
10.1 Contexte

Nous disposons potentiellement à ce stade d'une première modélisation 2D de chacun des niveaux du bâtiment étudié [32]. Les modèles 2D obtenus ne sont cependant pas référencés dans le même repère : il n'est donc pas possible de les superposer directement afin d'obtenir une modélisation 3D du bâtiment complet. Pour atteindre ce but, nous devons préalablement recaler les différents modèles 2D issus de chacun des niveaux. Ce recalage, qui est décrit en détail chapitre 12, permet de calculer la transformation à appliquer entre deux modèles de niveaux consécutifs afin de les aligner. La transformation est basée sur la mise en correspondance d'indices extraits de chacun des modèles 2D à considérer à partir des symboles architecturaux qui y sont présents. Nous décrivons dans ce chapitre les étapes mise en œuvre pour localiser un type d'indice intéressant pour le recalage à effectuer, les cages d'escalier.

En effet, parmi les différents éléments architecturaux à notre disposition sur la majorité des plans, certains apparaissent comme plus robustes et ainsi plus intéressants que les autres. C'est le cas des cages d'escalier, que l'on trouve inévitablement sur les plans architecturaux des bâtiments comportant plusieurs niveaux. Les cages d'escaliers sont associées à certaines caractéristiques [FIG. 10.1] :

- Elles ne sont généralement présentes qu'en petit nombre sur un plan donné, ce qui permet de limiter les hypothèses de mise en correspondance pour cette catégorie de symboles.
- Elles constituent un symbole *transversal* au bâtiment. On les trouve donc localisées au même emplacement, par rapport à l'axe des X et à l'axe des Y, sur 2 plans de niveaux successifs selon le repère global du bâtiment.
- Les représentations du symbole associé aux cages d'escalier peuvent être très différentes, même si une certaine homogénéité se dégage : les marches de l'escalier sont représentées symboliquement

32. L'étape de fusion des différents symboles détectés, permettant d'obtenir ce modèle 2D, est décrite chapitre 11.



(a) Rez-de-chaussée. (b) Premier étage.

FIG. 10.1 – *Exemple d'escaliers de deux niveaux consécutifs d'un même bâtiment.*

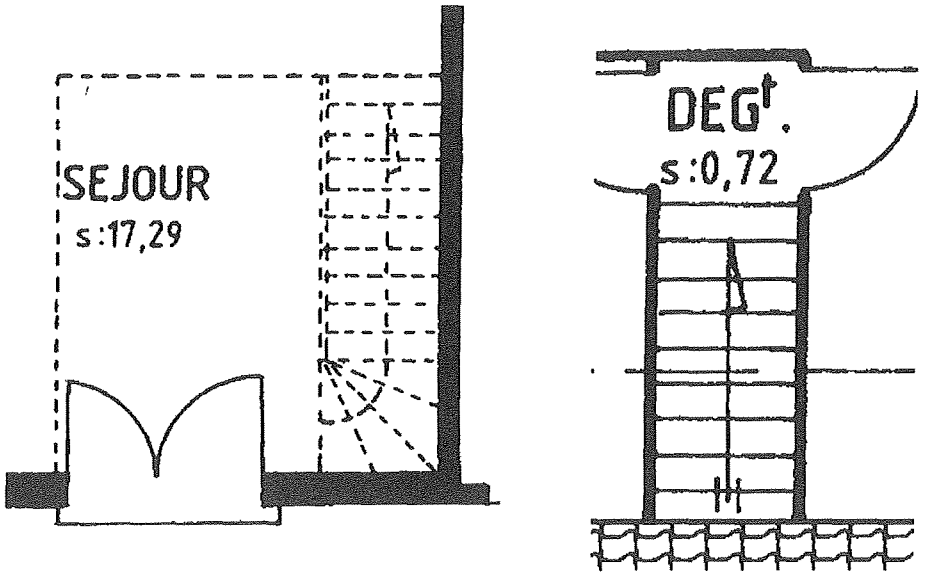
et en général une flèche superposée à la cage d'escalier permet de distinguer le sens de montée dans l'escalier. Cette dernière ajoute une difficulté à la compréhension lors de l'analyse du symbole, telle que celle évoquée lors de la segmentation texte / graphique [§ 3.3], caractéristique des problèmes de superposition de symboles sur les documents techniques.

- Enfin, si les représentations du symbole peuvent être très différentes en général, elles le sont aussi d'un niveau à l'autre [FIG. 10.1].

Les cages d'escalier présentent ainsi une invariance de localisation, mais pas de représentation, quoique cette représentation présente une structure régulière, assimilable à une sorte de texture par la répétition des marches composant l'escalier [FIG. 10.2]. Ce type de symbole est difficilement identifiable avec les méthodes de reconnaissance de graphiques décrites jusqu'à présent, mais peut être décelé comme un type particulier de texture. Aussi, afin de nous placer dans un cadre aussi général que possible, nous avons décidé de faire précéder la détection des cages d'escalier par une détection de textures [§ 10.2], cet agencement graphique entrant pour bonne part dans la représentation du symbole que nous recherchons. Nous présentons en particulier la méthode de Sánchez *et al.* [§ 10.3] qui est à la base de la détection que nous avons conçue [§ 10.4]. Nous effectuons ensuite la détection des escaliers [§ 10.5] à partir des résultats que nous obtenons avec notre détection de textures.

10.2 Quelques travaux en détection de textures

Outre les plans architecturaux, les textures se trouvent sur d'autres types de documents techniques, tels que les plans cadastraux ou les plans mécaniques. Ces textures ont généralement une connotation sémantique, apportant des informations sur le type de l'entité texturée, son utilité... Ainsi, la localisation et l'analyse de ces zones texturées est souvent une étape importante dans le processus de compréhension d'un document technique. Elle permet de manipuler des informations plus symboliques, et ainsi



(a) Rez-de-chaussée. (b) Premier étage.

FIG. 10.2 – *Autre exemple d'escaliers de deux niveaux consécutifs d'un même bâtiment. La représentation des marches présente une structure régulière, assimilable à une texture.*

d'avoir une abstraction supplémentaire dans l'analyse haut-niveau qui suit. Au niveau représentation, on distingue habituellement deux grandes catégories de textures :

- les textures *déterministes*, qui peuvent être décrites par un ou plusieurs motifs disposés selon des règles de disposition. Ces textures peuvent en outre être plus ou moins déformées, soit dans la forme des motifs qui les composent, soit dans leur disposition. Ce type de textures est utilisé par exemple pour la représentation d'échiquiers, de grilles, de hachures...
- les textures *stochastiques* qui sont composées de primitives disposées (semi-)aléatoirement, à l'image des points que l'on peut observer sur une télévision déréglée ou à la « texture » de certains matériaux, tissus...

Il existe également certaines textures qui se situent entre les catégories citées ci-dessus. C'est le cas de certaines textures déterministes à la base, dont la déformation du motif ou de la disposition est très marquée. Elles ne sont alors généralement plus classées dans cette catégorie, dont elles sont alors trop éloignées.

La segmentation de documents en zones texturées a fait l'objet d'un certain nombre de publications, présentant des méthodes qui envisagent ce problème sous différents angles, notamment en fonction du type de textures recherché. Van Gool *et al.* [Goo 85] proposent un état de l'art de ces méthodes en les classifiant. Ils différencient en particulier :

- les approches *structurelles* qui considèrent les textures comme des répétitions régulières d'éléments structurés et qui sont donc particulièrement bien adaptées à la recherche de textures déterministes ;

- les approches *statistiques* qui envisagent la détection de textures à partir d'un ensemble de mesures statistiques extraites localement sur toute l'image, et qui sont plus adaptées à la recherche de textures stochastiques.

Ces grandes familles de méthodes sont elles-mêmes composées d'un grand nombre de méthodes prenant en compte les spécificités de chaque type de textures, sous des angles différents et parfois complémentaires. La recherche dans ce domaine est toujours active et de nouvelles méthodes ont été proposées plus récemment. Ainsi, Tüceryan et Jain [Tüc 90] ont développé une méthode de segmentation de textures basée sur une décomposition de l'image en diagramme de Voronoï. Ils appliquent cette méthode sur des images présentant plusieurs types de textures et en extraient des régions de points, de symboles... Fu [Fu 83] décrit une approche syntaxique pour extraire les textures contenues dans une image. Enfin, les différentes approches sont parfois combinées. C'est le cas pour Unser [Uns 86] qui décrit une approche structurelle et statistique de la détection de textures, qu'il utilise pour discriminer les différents types de textures présents dans une image.

10.3 Méthode de Sánchez *et al*

Pour notre part, nous avons choisi une méthode développée par Sánchez *et al.* [Sán 97, Lla 98], basée sur une méthode proposée par Lam et Ip [Lam 94]. C'est une méthode de type structurelle utilisée en particulier pour l'analyse de textures sur des plans architecturaux, ce qui est exactement le type de plans que nous analysons. Les motifs composant les textures recherchées sont appelés, sous leur forme élémentaire, *texels* et correspondent dans ce cas à des polygones. La méthode effectue ainsi une recherche d'agencement régulier de polygones similaires dans les images vectorisées des plans architecturaux.

Afin d'extraire les différents polygones présents dans une image vectorisée donnée, une méthode proposée par Jiang et Bunke [Jia 93] est utilisée. Cette méthode est utilisée dans l'algorithme [ALG. 14] présenté plus loin. Les polygones extraits, correspondants à des séquences closes de coins, sont représentés par une chaîne cyclique de la forme $P = c_1, c_2, \ldots, c_n$. La détection de textures en elle-même est ensuite lancée. Pour cette détection, Lam et Ip [Lam 94] utilisent une approche structurelle, considérant une zone texturée comme une composition d'éléments de texture, les *texels*, qui dans ce cas correspondent aux polygones que nous avons extraits.

L'idée est d'utiliser une structure de pyramide irrégulière pour représenter une segmentation d'image, le regroupement de textures s'apparentant alors à un problème de contraction de graphe [Mon 91]. Une pile de graphes, dont les nœuds correspondent à des regroupements de polygones et les arcs aux relations d'adjacence qui existent entre ces regroupements, est construite. Initialement, chaque polygone extrait précédemment est considéré comme un regroupement de polygones, composé d'un seul élément. Les relations de voisinage entre les regroupements correspondent aux relations d'*adjacence* qui existent entre les polygones, ce qui est le cas lorsque deux polygones partagent au moins un côté. Les polygones sont ensuite regroupés selon des critères de similitude. Les regroupements créés constituent les noeuds du graphe contenu dans le niveau suivant de la pyramide [FIG. 10.3]. Le processus est itéré jusqu'à ce qu'il ne soit plus possible de créer de nouveaux regroupements. La construction de la pyramide irrégulière s'effectue ainsi en partant du premier graphe et en appliquant l'algorithme de contraction de graphe jusqu'à l'obtention de la segmentation de l'image en zones texturées. Ces dernières constituent alors les nœuds du dernier graphe obtenu, sommet de la pyramide construite.

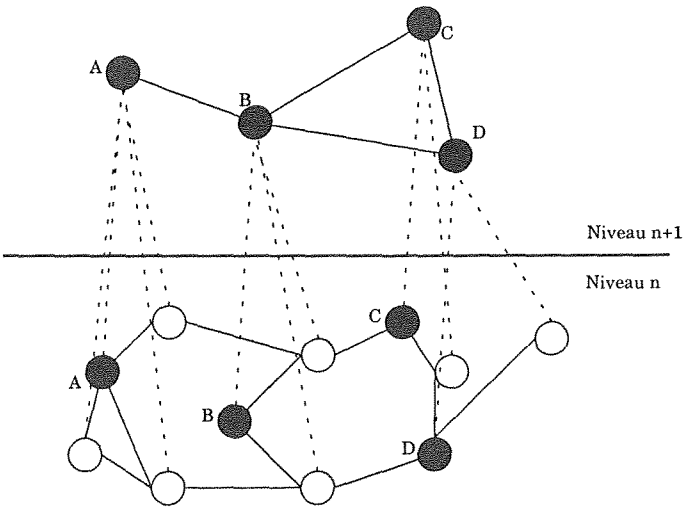


FIG. 10.3 – *Liens entre les nœuds père et fils entre deux niveaux de la pyramide des graphes. Les nœuds sont regroupés d'un niveau à l'autre autour de nœuds* survivants *(en noir).*

10.4 Description de notre détection de textures

À partir de la méthode d'extraction de polygones et de celle permettant de construire le premier niveau de la pyramide, nous avons développé un algorithme [ALG. 14]. Celui-ci accepte en paramètre la liste des vecteurs décrivant l'image sur laquelle nous travaillons, et complète au fur et à mesure un graphe dont les nœuds sont des ensembles de polygones et les arcs les relations d'adjacence qui existent entre les ensembles de polygones. De ce fait, les nœuds du graphe sont étiquetés par les ensembles qu'ils portent, et les arcs ne sont pas attribués, étant donné que leur simple existence est suffisante pour exprimer la relation qui existe entre deux nœuds. À partir du premier graphe obtenu, nous construisons une suite de graphes, chacun obtenu à partir du précédent, regroupant progressivement les nœuds pour déterminer les textures présentes sur le plan étudié. Ces graphes sont stockés dans une pyramide irrégulière permettant de représenter la hiérarchie de regroupements effectués, et finalement les textures détectées [FIG. 10.3].

Afin de comparer les polygones, pour mesurer leur similarité et les regrouper, il nous faut définir une distance entre eux qui permette de les comparer. L'expression des polygones grâce à des chaînes cycliques de la forme $P = c_1,c_2,\ldots,c_n$ permet l'utilisation de la distance d'édition de chaînes, telle qu'elle est utilisée par de nombreux auteurs [Mae 91, Tsa 84, Bun 93b]. Cette méthode comprend la définition de certaines opérations d'édition de chaînes, comme l'*insertion*, l'*élimination* ou encore la *substitution*, en leur associant un coût d'édition. La distance entre deux chaînes cycliques est alors exprimée par la composition de coût minimal de ces opérations pour passer d'une chaîne à l'autre. Cela revient intuitivement à définir cette distance à partir des différences de nombre de côtés des polygones, d'orientation et de longueur de ces côtés. Cette mesure est cependant mal adaptée au type de textures, correspondant aux marches d'escaliers, que nous voulons extraire des images. En effet, en reprenant les exemples de plans présentés précédemment, on distingue que la régularité de forme, ou de nombre de côtés, n'est pas l'une des caractéristiques des textures auxquelles nous nous intéressons, essentiellement à cause de la flèche superposée à la cage d'escalier [FIG. 10.1, 10.4]. Ce problème peut en partie être résolu en appliquant un prétraitement sur l'image vectorisée de départ. Ce prétraitement consiste à supprimer tous les segments possédant au moins une extrémité *libre*, c'est-à-dire non connectée à une autre primitive. Il doit être appliqué plusieurs fois, les segments supprimés lors d'une passe pouvant mettre à

ALG. 14 – GrapheInit(liste $lesSegments$, graphe<ensemble<Polygone>> $leGraphe$)

```
1: listeCoins ← Construction à partir de l'algorithme décrit dans [Jia 93]
2: listeCoins.Tri()
3: listeRelations ← ∅
4: tant que ¬listeCoins.Vide() faire
5:   unCoin ← Recherche du prochain coin inutilisé dans listeCoins
6:   listeCoins.Supprimer(unCoin)
7:   hypothèsePolygone ← ∅
8:   hypothèsePolygone.Ajout(unCoin.point(0))
9:   hypothèsePolygone.Ajout(unCoin.point(1))
10:  hypothèsePolygone.Ajout(unCoin.point(2))
11:  stop ← faux
12:  tant que ¬stop faire
13:    si il est possible de trouver un successeur coinSuc de unCoin dans listeCoins alors
14:      listeCoins.Supprimer(coinSuc)
15:      hypothèsePolygone.Ajout(coinSuc.point(2))
16:      si hypothèsePolygone.Premier() = hypothèsePolygone.Dernier() alors
17:        stop ← vrai
18:        unPolygone ← Création à partir de hypothèsePolygone
19:        unNœud ← leGraphe.AjoutNœud(ensembleVide().ajout(unPolygone))
20:        pour tout côté c de unPolygone faire
21:          listeRelations.Ajout(<c, unNœud>)
22:        fin pour
23:      sinon
24:        unCoin ← coinSuc
25:      fin si
26:    sinon
27:      stop ← vrai
28:    fin si
29:  fin tant que
30: fin tant que
31: listeRelations.TriSurCôtés()
32: pour tout couple d'éléments (c, n1) et (c, n2) de listeRelations faire
33:   leGraphe.AjoutArc(n1, n2)
34: fin pour
```

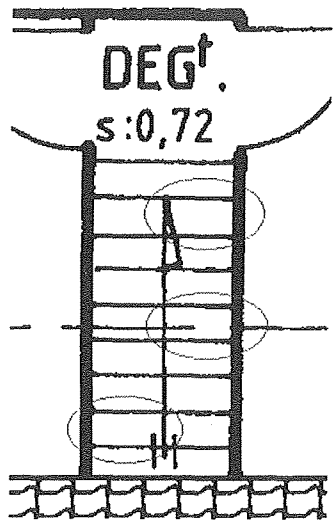


FIG. 10.4 – *Différentes formes de marches d'escalier rencontrées. Les zones encerclées mettent en particulier l'accent sur l'absence de régularité de forme des polygones représentant les marches de cet escalier.*

jour d'autres segments libres, jusqu'à ce qu'il n'y ait plus aucun segment supprimé lors d'une passe. Ce prétraitement résout en partie les difficultés présentées [FIG. 10.4], mais n'apporte pas pour autant une solution au type de textures présentée [FIG. 10.1].

Par contre, certaines caractéristiques communes se dégagent des différentes marches : l'adjacence et également une relative constance des aires des différentes marches, quoiqu'il convient de distinguer les marches sur lesquelles se trouve superposée la flèche des autres. Le premier critère, celui d'adjacence, est une caractéristique qui est prise directement en compte dans la méthode de détection de textures que nous utilisons. Nous ne le faisons donc pas ressortir dans le calcul de similitude de polygones que nous voulons effectuer. Le deuxième critère, qui porte sur la différence d'aire, n'est pas pris en compte directement dans la méthode sur laquelle nous nous basons. Nous l'utilisons donc pour définir un score S qui exprime la similitude relative d'aire entre les polygones. Ce score est défini ainsi :

$$S(P_1,P_2) = \frac{\min(\text{aire}(P_1),\text{aire}(P_2))}{\max(\text{aire}(P_1),\text{aire}(P_2))}.$$

Ce score est utilisé avec une large tolérance lors de la détection de textures, afin d'être sûr de pouvoir extraire toutes les zones texturées potentiellement candidates à la représentation d'une cage d'escaliers. La discrimination est alors effectuée lors de la détection de cages d'escaliers elle-même [§ 10.5].

Munis d'un score permettant de comparer les polygones entre eux et du graphe correspondant au premier niveau de notre pyramide, nous pouvons construire les niveaux suivants de la pyramide. La méthode de construction d'un niveau supplémentaire $n+1$ de la pyramide à partir du niveau n se décompose en plusieurs étapes [Lam 94], associées aux algorithmes correspondants que nous avons implantés :

1. initialisation et attribution d'un numéro identifiant pour chacun des nœuds du graphe [ALG. 15] ;
2. extraction des nœuds *survivants*, ayant le plus grand identificateur localement (par rapport à leurs voisins) [ALG. 16] ;

3. regroupement des nœuds non-survivants avec les nœuds survivants voisins si le score S associé à cette paire de nœuds est supérieur à un certain seuil. Dans le cas contraire, les nœuds non-survivants deviennent survivants à leur tour [ALG. 17] ;
4. création des nœuds de la couche $n+1$ à partir des survivants de la couche n [ALG. 18] ;
5. calcul des relations d'adjacence des nœuds de la couche $n+1$ à partir des relations de la couche n [FIG. 10.3] [ALG. 19] et [ALG. 20]. Pour cette étape, Lam et Ip définissent une règle selon laquelle s'il existe un chemin comportant au plus deux nœuds non survivants entre deux nœuds survivants de la couche n de la pyramide, les deux nœuds survivants sont connectés dans la couche $n+1$ de la pyramide. Cette règle est nécessaire : elle est une conséquence directe de la méthode de regroupement proposée.
   Elle n'est cependant pas suffisante. En effet, la recherche d'une relation d'adjacence peut alors impliquer jusqu'à 4 nœuds différents de la couche n, potentiellement inclus dans 4 nœuds différents de la couche $n+1$. Elle autorise dans ce cas la création de relations d'adjacence incorrectes, comme celle illustrée [FIG. 10.5]. Il est donc également nécessaire que les 4 nœuds de la couche n appartiennent à au plus 2 nœuds de la couche $n+1$.

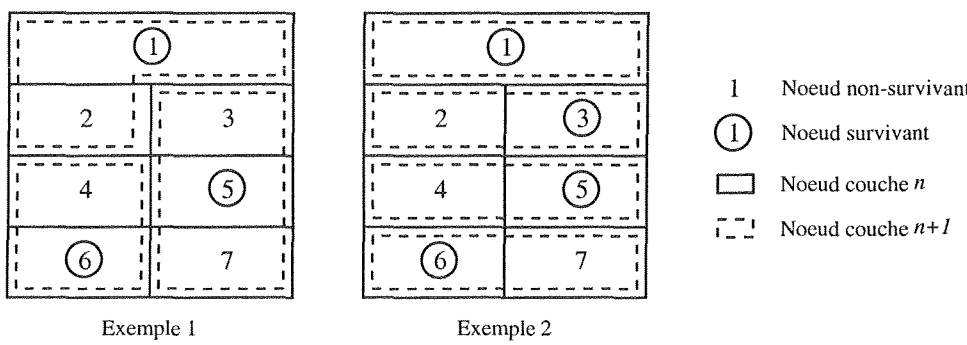


FIG. 10.5 – *Deux exemples de configuration avant le calcul des relations d'adjacence en ne tenant compte que de la règle énoncée par Lam et Ip. Sur le premier, le calcul des relations d'adjacence s'effectue sans problème. Sur le deuxième, le chemin 1-2-4-6 génère une mauvaise relation d'adjacence.*

ALG. 15 – GrapheTexture::Étape1

```
1: pour tout nœud n faire
2:     n.lien ← 0
3:     n.id ← pointeur sur n
4: fin pour
```

Quelques remarques sur les algorithmes proposés :

- étape 2 : la fonction `Nœud::EstSurvivant` correspond au test `(id = lien)` ;
- étape 3 :
  - le score calculé entre deux nœuds correspond au score calculé entre les moyennes d'aires des polygones regroupés dans ces deux nœuds ;
  - la valeur du seuil `SCOREMIN` est fixée expérimentalement (0,7 sur les exemples présentés) ;

ALG. 16 – GrapheTexture::Étape 2

```
1: encore ← vrai
2: tant que encore faire
3:    encore ← faux
4:    pour tout nœud n où ¬n.EstSurvivant() faire
5:       n.aléa ← random(0, 1)
6:    fin pour
7:    pour tout nœud n où ¬n.EstSurvivant() faire
8:       maxAléa ← maximum entre n.Aléa() et na.Aléa(), na ∈ Voisins(n)
9:       si n.Aléa() = maxAléa alors
10:         n.lien ← n.Id()
11:      fin si
12:   fin pour
13:   pour tout nœud n faire
14:      si n n'est pas survivant et qu'aucun de ses voisins ne l'est alors
15:         encore ← vrai
16:      fin si
17:   fin pour
18: fin tant que
```

ALG. 17 – GrapheTexture::Étape 3

```
1: pour tout nœud n où ¬n.EstSurvivant() faire
2:    max ← 0
3:    pour tout nœud na adjacent à n où na.EstSurvivant() faire
4:       si n.Score(na) >max alors
5:          max ← n.Score(na)
6:          nœudMax ← na
7:       fin si
8:    fin pour
9:    n.lien ← nœudMax.Id() si max >SCOREMIN, n.Id() sinon
10: fin pour
```

ALG. 18 – GrapheTexture::Étape 4(GrapheTexture *nouveauGraphe*)

```
1: pour tout nœud n où n.Estsurvivant() faire
2:    unNœud ← nouveauGraphe.AjoutNœud(n)
3:    n.succ ← unNœud
4:    pour tout nœud na adjacent à n où na.Lien() = n.Id() faire
5:       unNœud.AjoutPolygones(na.Polygones())
6:    fin pour
7: fin pour
```

ALG. 19 – GrapheTexture::Étape 5(GrapheTexture *nouveauGraphe*)

```
1: pour tout nœud n où n.EstSurvivant() faire
2:    ChercheRelation(n, n, listeAdjacence, 0, 0)
3: fin pour
4: listeAdjacence.Tri()
5: listeAdjacence.Unique()
6: pour tout adjacence a de listeAdjacence faire
7:    nouveauGraphe.AjoutArc(a.N1(), a.N2())
8: fin pour
```

ALG. 20 – GrapheTexture::ChercheRelation(Nœud *ref*, Nœud *dernier*, liste<Adjacence> *listeAdjacence*, entier *appel*, entier *chgmt*)

```
1: pour tout nœud n adjacent à dernier faire
2:    nouvChgmt ← chgmt + 1 si dernier.Lien() ≠ n.Lien(), chgmt sinon
3:    si nouvChgmt <1 et ref ≠ n alors
4:       si n.EstSurvivant() alors
5:          listeAdjacence.Ajout(<n.Succ(), ref.Succ()>)
6:       sinon
7:          si appel ≠ 2 alors
8:             ChercheRelation(ref, n, listeAdjacence, appel + 1, nouvChgmt)
9:          fin si
10:      fin si
11:   fin si
12: fin pour
```

- étape 5 :
  - le tri effectué sur la liste temporaire des relations d'adjacence prend en compte le premier nœud comme première clé et le second nœud comme seconde clé ;
  - la fonction `Liste::Unique` permet de ne garder qu'une occurrence d'un groupe d'éléments consécutifs de même valeur.

Cette série d'algorithmes est itérée jusqu'à ce que le graphe obtenu à la couche $n+1$ de la pyramide soit identique à celui de la couche n. Les nœuds de la dernière couche de la pyramide correspondent alors aux différentes zones texturées issues de la segmentation. Nous présentons [FIG. 10.6], les textures détectées à partir d'un exemple, dont celles correspondant aux escaliers. Les autres textures ne nous intéressent pas *a priori* sur les exemples présentés et dans notre contexte. Elles peuvent néanmoins être utilisées pour détecter des tuiles, du carrelage, des murs porteurs...

10.5 Recherche des escaliers à partir des textures détectées

À partir des textures détectées, nous devons maintenant distinguer les textures qui représentent des escaliers de celles qui représentent d'autres symboles architecturaux. La détection de textures a été implémentée comme une méthode générique et il est nécessaire d'injecter des connaissances *a priori* afin de discriminer les symboles recherchés. Ces connaissances peuvent être soit absolues, en décrivant de manière précise les règles de modélisation des escaliers, ou relatives, en énonçant des propriétés discriminatoires permettant de distinguer les escaliers des autres zones texturées détectées [FIG. 10.7].

En fait, la représentation des escaliers, comme celle des autres éléments architecturaux, n'est pas normée, mais se veut généralement assez fidèle à la réalisation sur le chantier. On retrouve ainsi fréquemment des courbures, ou des parties cachées lorsque des combles sont aménagés [FIG. 10.1] ; les marches de l'escalier sont généralement représentées. De ce fait, la modélisation des escaliers est extrêmement délicate tant les représentations sont différentes d'un ensemble de plans à l'autre, voire sur un même plan. En fait, l'une des seules caractéristiques exploitable pour distinguer les textures représentant les cages d'escaliers des autres textures est de considérer les marches de l'escalier. Leur forme n'est pas plus facile à modéliser que celle de l'escalier lui-même, pour les raisons évoquées ci-dessus et au regard des résultats fournis par la vectorisation, mais leur quantité est généralement constante : entre 5 et 15 marches suivant les configurations. En considérant que la cage d'escalier peut être recouverte par une flèche indiquant le sens de montée et que les marches sont dans ce cas dédoublées, de 5 à 30 éléments de texture sont requis pour la représentation de telles cages.

Nous proposons donc de nous baser sur cette seule caractéristique pour filtrer les résultats fournis par la détection de textures. La variété de représentation et le peu d'occurrence de ce type de symboles sur les plans architecturaux n'en font pas un bon candidat pour une modélisation, qui s'avérerait extrêmement coûteuse et peu rentable, voire pas du tout. En effet :

- sur les plans où peu de zones texturées sont présentes, le simple filtrage que nous proposons est souvent suffisant pour distinguer les cages d'escaliers. Il n'est donc pas nécessaire d'utiliser des contraintes supplémentaires pour faire cette distinction ;
- sur les plans où de nombreuses zones texturées sont présentes, comme les plans comportant des hachures, des représentations de toit ou de carrelage, les modélisations sont très difficiles à définir. En effet, de nombreux candidats sont extraits à chaque proposition de modèles, qui apparaissent alors comme inefficaces.

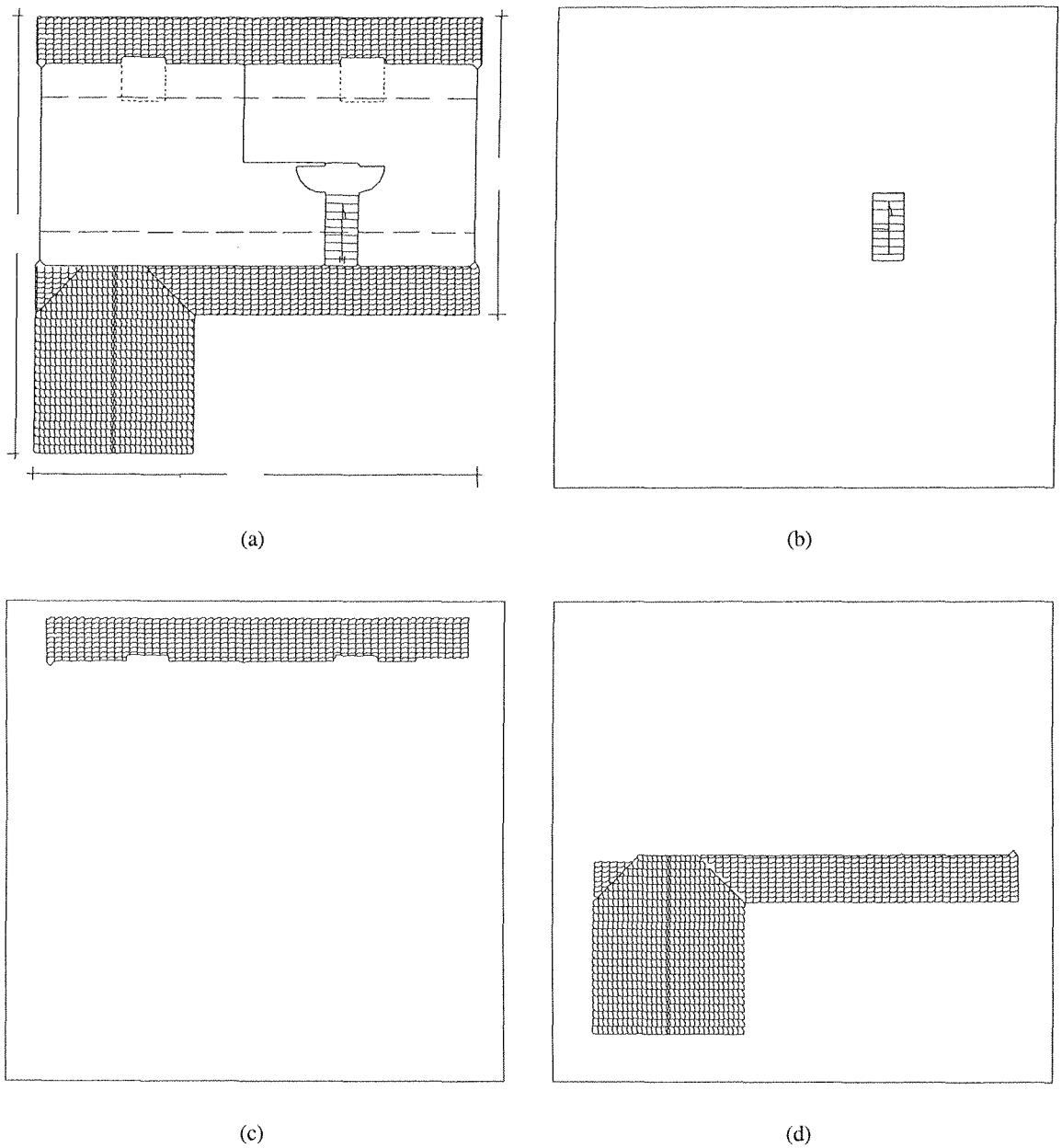
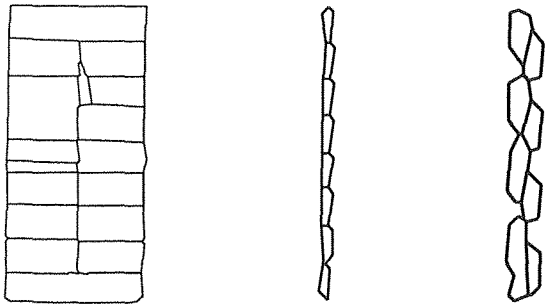


FIG. 10.6 – *Différentes textures détectées sur un plan architectural (a), dont celle correspondant aux escaliers (b). Les textures sont détectées à partir de leur représentation basée sur une structure répétitive et régulière.*



(a) (b) (c)

FIG. 10.7 – *Exemples de textures extraites. Les deux premières sont extraites à partir de* [FIG. 10.6(a)] *et la troisième à partir de* [FIG. 10.8(c)].

Nous pensons plutôt que la sélection finale est typiquement une des tâches qui doit être effectuée par le superviseur de l'analyse, selon les principes développés chapitre 13. Ce choix est *a priori* raisonnable, au vu de la difficulté de modélisation du symbole et du faible nombre de propositions que cela représente. Nous proposons donc de filtrer les textures selon leur nombre d'éléments, en fixant un nombre minimum et maximum autorisé, et de proposer au superviseur les textures résultantes. Celui-ci a alors la possibilité d'éditer ces résultat, pour créer, modifier ou supprimer des propositions, au moyen de l'interface graphique développée [§ 15.6]. Les bornes inférieure et supérieure du nombre d'éléments autorisés sont fixées par défaut aux valeurs données ci-dessus (respectivement 5 et 15).

Nous présentons [FIG. 10.8, 10.9] les résultats que nous obtenons à l'issue de notre détection d'escaliers, tels qu'ils sont présentés au superviseur. En général, il y a peu de zones texturées sur les plans architecturaux, et ainsi peu de candidats résultants de notre détection. Certains résultats sont insolites, comme la baignoire retenue [FIG. 10.9(f)], mais sont la marque de la difficulté de modélisation rencontrée pour cette détection. Bien sûr, il serait possible de supprimer ce type de résultat en injectant des connaissances *a priori* supplémentaires, mais l'intérêt de cette démarche est relativement faible en regard de la marginalité de ce type de résultat.

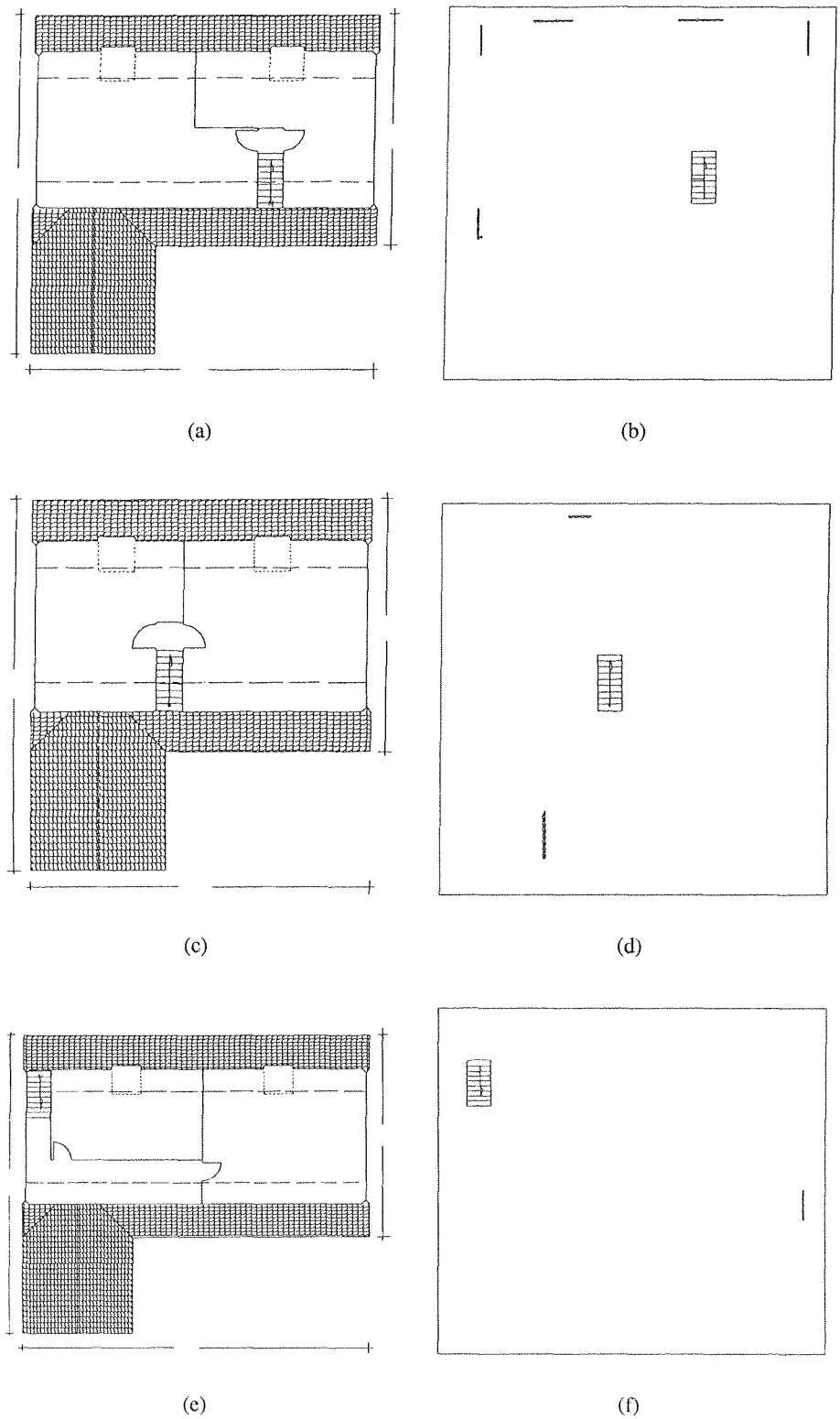


FIG. 10.8 – *Résultats obtenus sur une première famille de pavillons.*

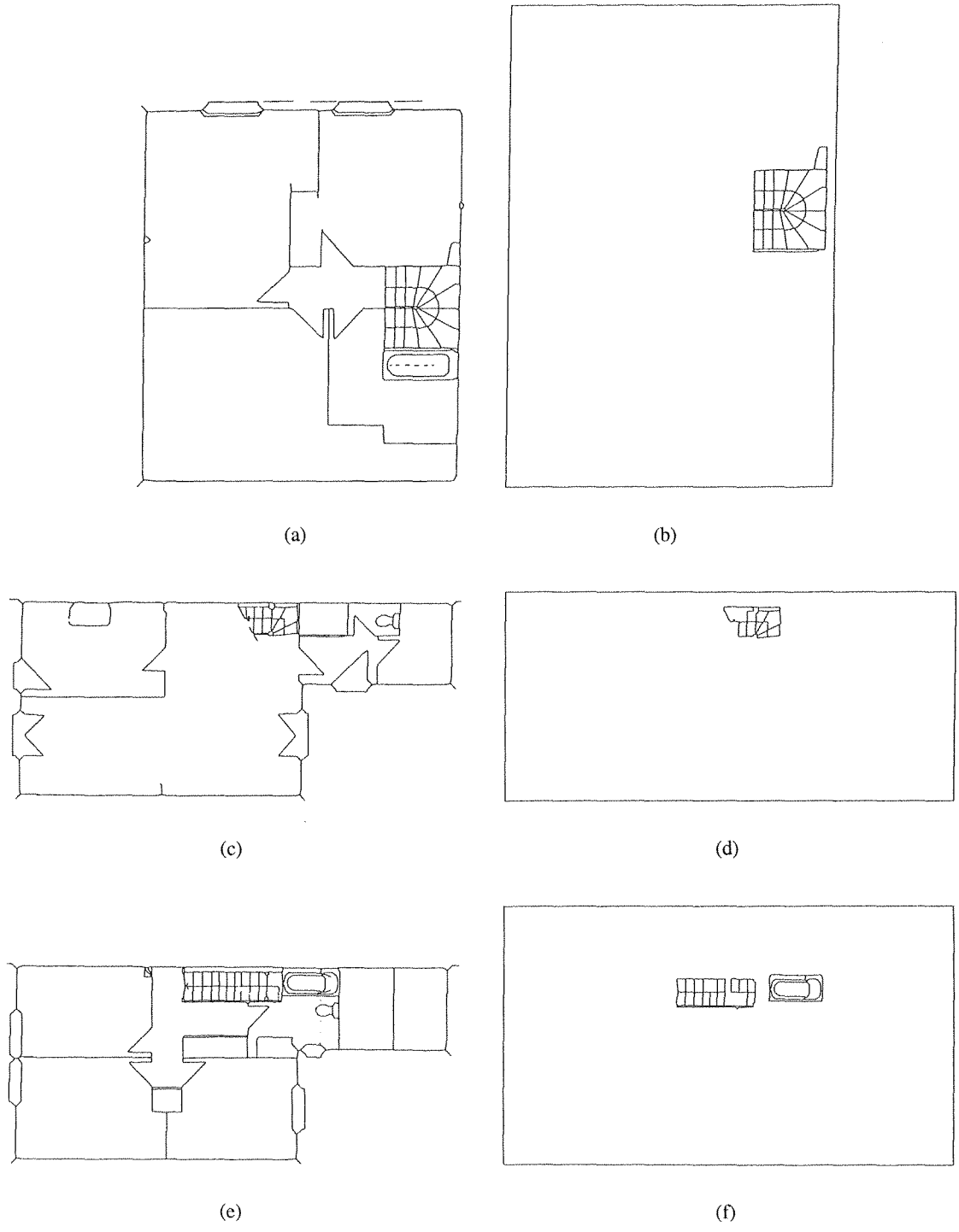


FIG. 10.9 – *Résultats obtenus sur une seconde famille de pavillons.*

Chapitre 11

Construction du modèle 3D d'un étage

11.1 Fusion des informations

Les phases d'analyse précédentes nous ont permis d'identifier les symboles architecturaux que nous recherchions. Dans notre présentation, l'enchaînement des différents traitements mis en œuvre est adapté au type de plans que nous avons traité. Cependant, de part la nature indépendante de ces traitements, l'enchaînement peut être modifié pour s'adapter à d'autres types de représentation. Quelle que soit la situation considérée, les symboles identifiés à ce stade sont localisés dans plusieurs images, résultant des traitements réalisés. Pour obtenir un modèle 2D unique regroupant ces informations, il est nécessaire de fusionner les différentes informations symboliques représentées dans ces images.

On peut distinguer à ce stade deux catégories différentes d'information. La première est celle des symboles déjà attribués par la catégorie de symboles à laquelle ils appartiennent. En effet, certains symboles architecturaux sont directement identifiés par des phases spécifiques. C'est le cas dans notre contexte des portes, des fenêtres ou des escaliers. La seconde catégorie est celle des symboles qui correspondent encore à ce stade à de simples primitives graphiques. C'est le cas des murs porteurs, qui correspondent à la couche vectorisée de traits forts, ou des cloisons, qui correspondent aux segments non utilisés de la couche vectorisée de traits fins. Il est nécessaire d'effectuer une opération de conversion pour ces derniers symboles, pour les attribuer par la catégorie de symboles à laquelle ils appartiennent. Afin de prendre en compte les évolutions ou les différences de représentation des plans que nous traitons, il est ainsi possible lors de l'étape de fusion d'attribuer n'importe quel type de primitives par n'importe quelle catégorie de symboles. Par conséquent, le principe du processus de fusion d'informations se résume au schéma présenté [FIG. 11.1].

Le processus de fusion s'effectue très simplement : les différentes informations étant positionnées par rapport au repère du plan original, ce processus correspond à un simple regroupement des informations au sein d'une même image, ce qui donne d'assez bons résultats. Le modèle 2D obtenu n'est cependant pas toujours exempt de défauts, notamment au niveau des jonctions entre les différents éléments architecturaux. Outre les artefacts introduits lors de la vectorisation, certains des traitements appliqués génèrent des imprécisions dans leurs résultats. C'est en particulier le cas lors de la reconnaissance de

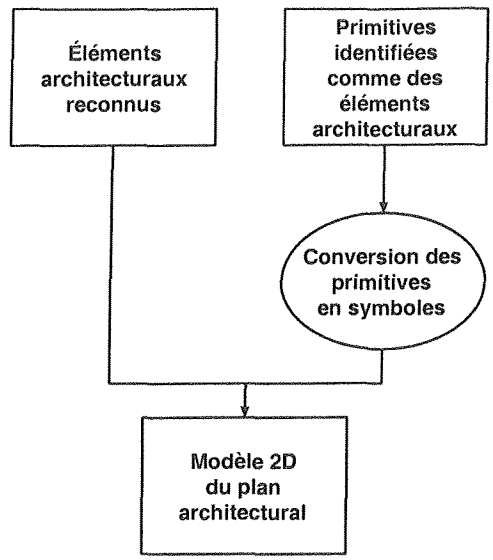


FIG. 11.1 – *Principe du processus de fusion des informations.*

symboles (chapitre 9). En effet, lorsque les symboles reconnus sont substitués aux primitives graphiques les représentant, le positionnement du symbole créé peut être imprécis. Au stade de la fusion des informations, il est alors possible que la connexité globale du modèle 2D, c'est-à-dire la connexité au niveau des différents symboles architecturaux qui le composent, soit affectée.

Si ce problème est délicat à résoudre au niveau même des différentes phases incriminées, il peut être partiellement résolu au niveau de la fusion. Une des solutions est de mettre en œuvre un traitement s'apparentant à une fermeture de contours, reliant les symboles architecturaux dont l'éloignement est inférieur à un certain seuil. Une autre possibilité est de faire intervenir l'utilisateur, qui a la possibilité de recaler les différents symboles à partir de l'interface interactive présentée chapitre 15. Dans tous les cas, une fois que la connexité globale du modèle est établie, il est aisé de calculer l'enveloppe englobante du niveau étudié pour déterminer les dimensions de sa dalle. Cette information est notamment utile lors des phases d'empilement des différents modèles et de la construction du modèle 3D du bâtiment complet.

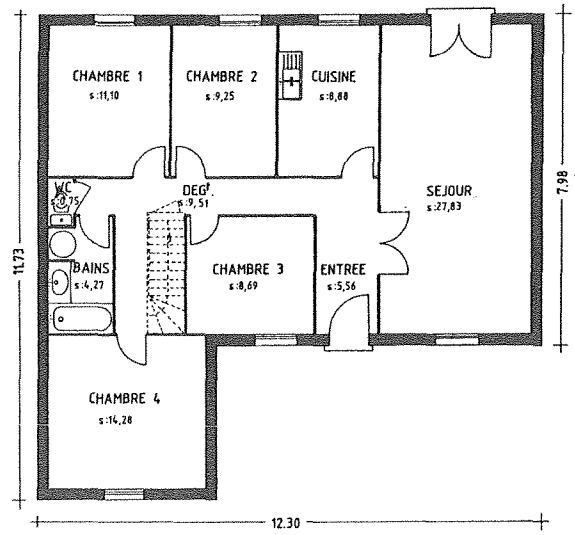
11.2 Élévation du modèle 2D obtenu

Suite à la fusion des informations extraites à partir du plan d'un niveau, nous disposons d'une description du niveau considéré en termes de symboles architecturaux. Nous supportons actuellement sept symboles différents : les fenêtres simples, les portes-fenêtres, les portes, les murs porteurs, les cloisons, les escaliers et les symboles transversaux de différentes natures (conduites, colonnes sèches, etc.), que nous désignons tous par le terme générique *conduites* dans la suite de ce rapport à des fins de simplification. La description du niveau selon ces symboles constitue un modèle 2D. Bien entendu, la structure modulaire adoptée dans notre analyse permet d'intégrer facilement de nouveaux symboles sans remettre en cause l'existant.

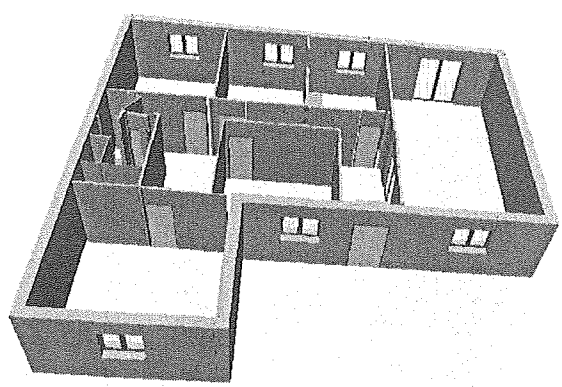
Afin d'obtenir un modèle 3D du même plan à partir du modèle 2D calculé, plusieurs possibilités sont envisageables. La première d'entre elles consiste à exploiter les cotations parfois présentes sur les plans. En effet, certaines de ces cotations fournissent des informations sur la hauteur des composantes

architecturales. Une analyse automatique de ces cotations, à partir d'une des méthodes évoquées [§ 1.1], permet alors d'associer à chaque composante la hauteur à lui affecter. Nous n'avons cependant pas retenu cette solution : ce type de cotation est assez rare sur les plans étudiés, et n'est en fait généralement disponible que sur des plans en coupe. Cela nécessite dans ce cas de considérer plusieurs plans pour un niveau donné et de fusionner des informations récoltées sur chacun des plans. La mise en œuvre de cette technique est donc assez importante, alors qu'il y a finalement peu d'information à extraire — si nous nous en tenons à la recherche des cotations. En effet, sur la majeure partie des plans existants, tous les éléments architecturaux comme les murs, les cloisons, les escaliers, les conduites, présentent tous la même hauteur pour un niveau donné. Il n'y a guère que dans les plans de bâtiments plus atypiques, comme ceux des salles de cinéma, des grands bâtiments publics, que cette information peut varier d'une composante à l'autre.

Une autre solution, que nous avons retenue pour notre système, est d'appliquer une simple *extrusion* sur les éléments architecturaux. Pour cela, une élévation, qui est globale pour le moment, est associée à chacun des symboles architecturaux. La valeur du paramètre d'extrusion est fournie par l'utilisateur pour chacun des niveaux du bâtiment considéré. Cette étape permet d'obtenir un modèle 3D d'un plan donné à partir de son modèle 2D. Pour nos besoins, cette approche permet d'obtenir des résultats satisfaisants. La figure [FIG. 11.2] présente un plan architectural et le modèle 3D obtenu à partir de ce plan.



(a) Plan architectural.



(b) Reconstruction 3D.

FIG. 11.2 – *Exemple de reconstruction 3D obtenu après extrusion.*

Chapitre 12

Mise en correspondance de niveaux

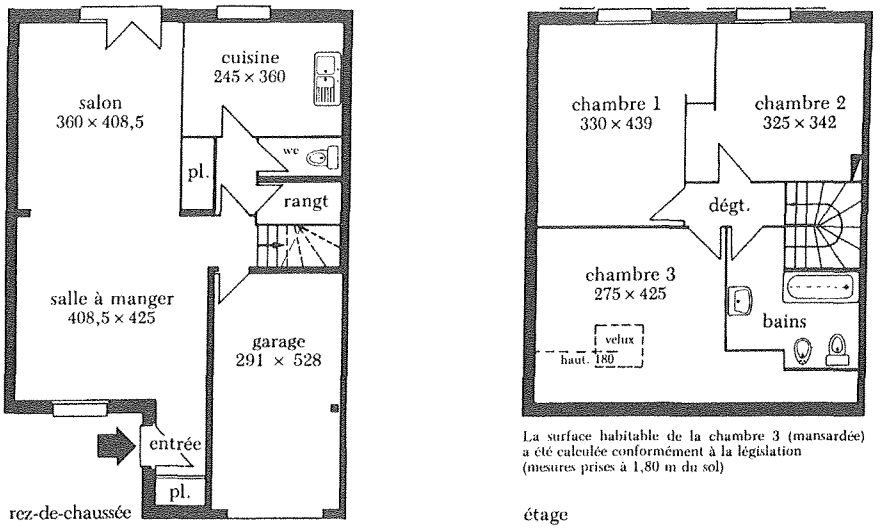
12.1 Introduction

Nous avons présenté dans les chapitres précédents les méthodes mises en œuvre pour obtenir un modèle 3D d'un niveau de bâtiment à partir des entités architecturales reconnues sur le plan de ce niveau. Pour construire un modèle d'un bâtiment complet, ces méthodes sont tout d'abord appliquées sur le plan de chaque niveau du bâtiment. Le modèle 3D complet du bâtiment est alors obtenu en empilant les uns sur les autres les modèles 3D de chacun de ses niveaux. La manière dont deux niveaux consécutifs s'ajustent est déterminée au moyen d'un appariement d'indices sélectionnés à partir des modèles 3D de chaque niveau. Cet appariement permet de calculer la transformation géométrique à appliquer à la structure d'un niveau pour le recaler sur celui qui le précède. Ce chapitre présente les différentes étapes composant cette opération de recalage [Dos 99b, Dos 00a].

12.2 Indices nécessaires à la mise en correspondance

Afin d'effectuer une mise en correspondance de deux niveaux consécutifs, il est nécessaire de se baser sur des indices robustes transversaux au bâtiment, c'est-à-dire alignés dans la réalité selon l'axe vertical. Après la mise en correspondance des indices extraits sur chaque niveau, il est alors facile de calculer la transformation géométrique qui permet de recaler ces niveaux. La première étape consiste donc à rechercher les indices présents à chaque niveau parmi les symboles architecturaux de la structure 3D du niveau considéré.

De nombreux plans architecturaux ont été préalablement étudiés afin de déterminer le type des indices à prendre en compte pour la mise en correspondance. On remarque par exemple sur la figure [FIG. 12.1] que la forme des escaliers et la forme globale du bâtiment sont différentes d'un niveau à



(a) Premier niveau. (b) Deuxième niveau.

FIG. 12.1 – *Plans architecturaux de deux niveaux d'un pavillon.*

l'autre du bâtiment. Sur la figure [FIG. 12.2], au contraire, ils sont relativement similaires. Cependant, sur cette même figure, il n'y a qu'un petit nombre d'indices disponibles : la plupart des murs porteurs (correspondant aux traits forts) sont masqués par la texture représentant le toit.

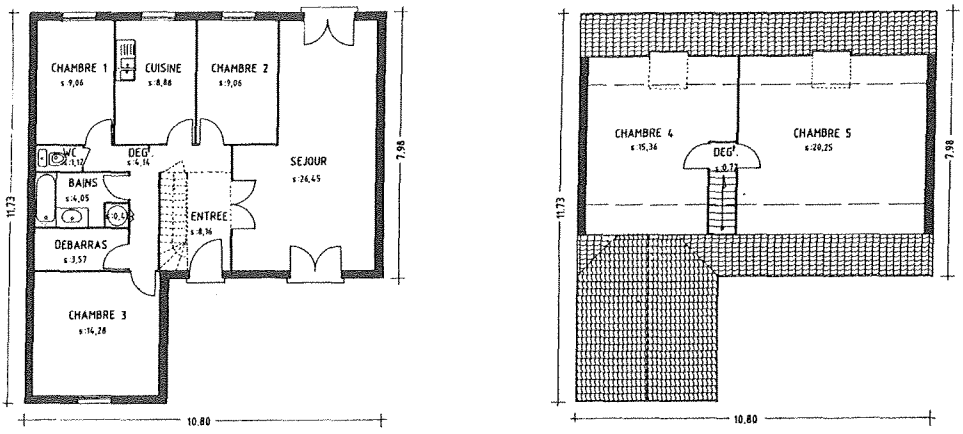
D'un côté, il apparaît que les symboles tels que les escaliers ou les conduites peuvent être utilisés directement comme indices car ce sont des symboles *transversaux*. Leur position est ainsi généralement invariante entre deux niveaux consécutifs. De plus, s'ils sont présents à un niveau donné, on les retrouve sur un niveau consécutif (inférieur ou supérieur). D'un autre côté, la mise en correspondance ne peut pas être exclusivement basée sur de tels symboles étant donné qu'ils sont peu nombreux, voire, dans le cas des conduites, souvent absents. Il faut de plus considérer que la forme des escaliers peut changer d'un niveau à l'autre [FIG. 12.1], ce qui ne permet pas une grande précision pour la mise en correspondance.

Les cloisons ne constituent pas des indices intéressants car leur positionnement n'est pas forcément identique entre les différents niveaux. Ce n'est pas le cas avec les murs porteurs quand la forme extérieure du bâtiment considéré est relativement similaire pour chaque niveau. Ils peuvent ainsi être considérés comme indices pertinents, et plus particulièrement encore aux *coins* correspondant à l'intersection de deux de ces murs, ce point représentant un indice dont la localisation est très précise.

Quand il n'y a ni coin ni conduite sur les plans étudiés, comme sur la figure [FIG. 12.2(b)], seuls les murs porteurs, *i.e* de simples segments, sont disponibles. Dans ce cas, la mise en correspondance ne peut être très précise, vu que de tels segments ne fournissent pas de points caractéristiques pouvant être utilisés comme « ancres » pour l'appariement.

Quatre catégories d'indices ont finalement été retenus :

- **Les coins** : ce sont des indices pertinents quand la forme extérieure du bâtiment est relativement stable. Les segments déterminant les coins sont orientés, ce qui permet d'attribuer l'angle qu'ils forment par une orientation, en se basant par exemple sur la pente de la bissectrice.



(a) Premier niveau.

(b) Deuxième niveau.

FIG. 12.2 – *Plans architecturaux de deux niveaux d'un autre pavillon.*

Les coins sont déterminés grâce à une méthode décrite par Jiang et Bunke [Jia 93]. Parmi toutes les paires de segments connectés correspondant aux murs porteurs, les coins sont supposés former des angles proches de $\frac{\pi}{2}$ radians. Ceci constitue une hypothèse *a priori*, quoique cela ne soit par forcément vrai, surtout dans l'architecture moderne.

- **Les escaliers** : ils sont systématiquement présents sur les plans des bâtiments à plusieurs étages, quoiqu'ils puissent se présenter sous différentes formes sur chacun d'eux. Certains escaliers incluent une flèche permettant de signaliser le sens de la montée. Cette information devrait être utile pour la mise en correspondance mais, en pratique, la reconnaissance de telles flèches échoue souvent. La raison principale de cet échec est liée au problème général de la superposition de symboles, la complexité de représentation des escaliers accentuant cet échec.
- **Les conduites** : leur forme est toujours invariante et le plus souvent symétrique (carré ou cercle). Leur mise en correspondance doit donc être définie à cette rotation près.
- **Les murs porteurs** : comme nous l'avons précédemment énoncé, ces indices sont orientés et, par définition, leur position est généralement invariante. Cependant, il y a au moins deux raisons pour lesquelles ils ne constituent pas des indices de confiance.
  - Premièrement, la disposition des menuiseries (portes et fenêtres) peut différer d'un niveau à l'autre. Cela ne change rien au niveau du bâtiment, qui conserve des murs sur toutes les longueurs de ses pans de murs. En revanche, le plan correspondant ne représente plus les murs aux mêmes endroits — étant donné qu'il y a des interruptions pour permettre la représentation des menuiseries —, ce qui complique les mises en correspondance.
  - Deuxièmement, un mur reconnu comme un simple segment à un niveau donné peut être représenté par plusieurs segments sur un autre niveau en raison du bruit contenu sur l'image initiale qui altère le résultat obtenu après vectorisation. Un indice unique peut dans ce cas avoir plusieurs correspondants et *vice versa*. Tout ceci va compliquer la mise en correspondance et diminuer la qualité des résultats obtenus.

Le tableau [TAB. 12.1] présente un état récapitulatif des caractéristiques des indices recensés. Il est difficile d'affecter à ce stade un taux de confiance en regard de chaque catégorie d'indices. En effet, aucun

	Symbole transversal	Invariance de position	Invariance de forme	Possibilité d'orientation	Disponibilité
Coins	0	+	+	+	0
Murs porteurs	0	+	-	+	+
Conduites	+	+	+	0	-
Escaliers	+	0	-	-	0

TAB. 12.1 – *Caractéristiques des indices recensés. Légende pour une caractéristique donnée : (+) bonne adéquation, (0) adéquation moyenne, (-) mauvaise adéquation.*

des indices retenus ne cumule toutes les qualités nécessaires à une mise en correspondance robuste et ce taux dépend beaucoup des plans étudiés.

12.3 Choix d'une méthode de mise en correspondance

Nous devons maintenant déterminer une méthode permettant de mettre en correspondance les indices. Le choix de cette méthode dépend de certains critères induits par les observations précédentes :

- Comme les plans peuvent être dessinés à des échelles différentes à partir de sources séparées, la mise en correspondance des indices nécessite le calcul de la transformation, c'est-à-dire une combinaison d'une translation, d'une rotation et d'une échelle, qui permet d'effectuer le recalage de deux niveaux entre eux.
- L'ensemble des indices disponibles à chacun des niveaux est généralement très disparate. Il n'y a guère que les symboles que nous appelons transversaux qui sont stables (en nombre et en position). Leur position (relative) ne change pas, même lorsque la forme extérieure des niveaux n'est pas homogène. La position relative est également invariante à la translation, à la rotation et au changement d'échelle. Il semble alors approprié de représenter un niveau comme un modèle relationnel : un indice est caractérisé par la position relative d'un nombre minimum $\mathcal{V}_{\min}$ d'indices voisins.
- Il y a relativement peu d'indices à mettre en correspondance et chaque indice est associé à un petit nombre d'attributs simples. Un algorithme d'appariement sophistiqué apparaît ainsi excessif par rapport à la complexité du problème. Bien que les ensembles d'indices aient seulement quelques éléments en commun, nous n'avons pas besoin d'une technique telle qu'un isomorphisme de (sous-)graphes inexact, qui de plus est très coûteuse en temps. Nous avons aussi rejeté les algorithmes basés sur la relaxation discrète à cause de leur manque de flexibilité. Ils s'exécutent comme des processus globaux et il est difficile d'avoir le contrôle pendant le déroulement de leurs différentes étapes.

En fait, notre problème semble très proche de celui présenté par Bolles et Cain [Bol 83] pour la reconnaissance de charnières dans une poubelle : les objets étudiés sont représentés par de simples caractéristiques (des trous et des cales) ainsi que par leurs positions relatives, les uns par rapport aux autres. Les hypothèses d'appariement entre les indices du modèle d'une charnière et les indices extraits de l'image de la poubelle sont représentés par des nœuds dans un graphe de compatibilité. Dans ce graphe, un arc entre deux nœuds exprime la consistance existant entre les deux hypothèses, et la plus grande clique maximale, c'est-à-dire le plus grand ensemble de nœuds mutuellement connectés, représente la meilleure mise en correspondance entre le modèle et l'image.

Cette méthode présente de nombreux avantages. Tout d'abord, elle repose sur l'étude des positions relatives des différents indices entre eux, et est ainsi insensible au type de transformation dont nous devons tenir compte. D'autre part, l'utilisation d'un graphe de compatibilité, et l'extraction de la plus grande clique maximale de ce graphe, permet de traiter des couples de niveaux dont les ensembles d'indices peuvent être dissimilaires. La méthode extrait ainsi la meilleure solution en fonction des indices disponibles. La qualité des résultats présentés en reconnaissance de charnières nous a convaincu d'utiliser une approche similaire pour nos besoins.

12.4 Mise en œuvre d'une méthode à base de cliques maximales

12.4.1 Choix des indices

Une mise en correspondance fiable entre une paire de niveaux donnés, $\mathcal{N}_1$ et $\mathcal{N}_2$, doit porter sur un nombre minimum d'indices robustes. Cependant, suivant les plans, les catégories d'indices n'ont pas toujours la même pertinence. Un ordre de priorité, correspondant à un taux de confiance, est donc associé à chaque catégorie d'indices. Cet ordre de priorité est utilisé pour déterminer dans quel ordre sont considérés les indices, ce qui permet de s'intéresser aux indices les plus pertinents en priorité.

Un ordre de priorité par défaut est défini sur les catégories d'indices, dépendant de leur invariance de forme et de position, à partir de la synthèse du paragraphe [§ 12.2] : les conduites, les escaliers, les coins et les murs porteurs, dans cet ordre décroissant. La stabilité de la position prévaut sur la stabilité de forme étant donné que cette première permet d'effectuer des mises en correspondance plus précises, et ainsi un calcul plus précis de la transformation liant $\mathcal{N}_1$ et $\mathcal{N}_2$. Cet ordre par défaut est représentatif des configurations les plus fréquemment observées, et peut être changé avant le début de l'étape de mise en correspondance. Il serait d'ailleurs intéressant de déterminer des heuristiques permettant d'affiner automatiquement cet ordre de priorité ; la configuration reste actuellement manuelle. On considère ensuite :

- $n_{\min}$, le nombre minimum d'indices nécessaires pour effectuer un appariement robuste. Sa valeur est déterminée expérimentalement.
- $c^i_{\min}$ le nombre minimum d'indices appartenant à la catégorie $\mathcal{C}_i$ disponible dans $\mathcal{N}_1$ et $\mathcal{N}_2$. Cela correspond au nombre maximum d'appariements consistants qui peuvent être réalisés dans cette catégorie.

Les catégories $\mathcal{C}_i$ sont successivement considérées dans leur ordre de priorité, en additionnant leur $c^i_{\min}$ correspondant. Lorsque le total atteint $n_{\min}$, les catégories potentiellement restantes sont ignorées et les autres sont sélectionnées pour le processus de mise en correspondance globale. De tels principes garantissent que les indices bruités ou non robustes ne seront pas utilisés si l'ensemble d'indices pertinents est assez grand, et ainsi ne perturberont pas la mise en correspondance. Les indices voisins utilisés pour caractériser un indice donné sont sélectionnés à partir des catégories retenues.

12.4.2 Le graphe de compatibilité

Le graphe de compatibilité est la structure de données constituant le cœur de la méthode des cliques maximales [§ 8.3.3]. Une hypothèse de mise en correspondance entre un indice retenu du niveau $\mathcal{N}_1$ et un indice retenu du niveau $\mathcal{N}_2$ constitue un nœud dans le graphe de compatibilité. Pour cette étape, chaque indice est assimilé à un point : le centre de gravité du polygone représentant les contours d'un escalier ou

d'une conduite, l'intersection de deux segments formant un coin, le milieu du segment représentant un mur porteur. L'orientation relative de deux indices est calculée comme la direction du segment joignant leur point représentatif.

Deux indices $i_1 \in \mathcal{N}_1$ et $i_2 \in \mathcal{N}_2$ sont associés pour former une hypothèse de mise en correspondance à la condition que leur taux de confiance $\Delta(i_1,i_2)$ soit inférieur à un seuil fixé. La fonction Δ est définie comme la somme des distances minimums δ entre un voisin du premier indice et les voisins du second indice :

$$\Delta(i_1,i_2) = \sum_{j=1}^{j=\nu_{\min}} \min_{k=1}^{k=\nu_{\min}} \delta(i_{1.j},i_{2.k}),$$

où $i_{p.q}$ représente le q-*ième* voisin de l'indice i_p. δ exprime la différence entre les orientations relatives des deux indices voisins considérés. Si ces indices n'appartiennent pas à la même catégorie, δ renvoie une valeur qui empêche tout appariement entre ces deux indices.

Une fois tous les nœuds du graphe construits, les arcs du graphes sont construits. Ces arcs expriment la compatibilité d'hypothèses entre les nœuds qu'ils relient. Soient $i_1^a{:}i_2^j$ et $i_1^b{:}i_2^k$ deux hypothèses de mise en correspondance. Un arc relie les deux nœuds correspondant dans le graphe s'ils représentent une hypothèse consistante, ce qui se traduit par :

- $a \neq b$ et $j \neq k$ (un même indice de $\mathcal{N}_1$ ne peut évidemment pas s'apparier avec deux indices différents de $\mathcal{N}_2$, et *vice versa*) ;
- $\theta(i_1^a) - \theta(i_2^j) \approx \theta(i_1^b) - \theta(i_2^k)$, où θ représente la direction absolue (dans le système de coordonnées du plan image) d'un indice : la direction de la bissectrice de l'angle formé par le coin et la direction du segment représentant le mur, respectivement. En d'autres termes, la rotation induite par le premier appariement doit être la même que celle induite par le second appariement. Remarque : cette contrainte n'est utilisée que sur les catégories d'indices orientables, c'est-à-dire les coins et les murs ; elle n'a pas de sens sur les indices qui ne sont pas orientables.

12.4.3 La plus grande clique maximale

La plus grande clique maximale, c'est-à-dire le plus grand sous-graphe complètement connecté, qui peut être extraite du graphe de compatibilité représente le meilleur appariement entre les deux niveaux traités. Le calcul de la plus grande clique maximale est un problème NP-complet bien connu [Gar 79] et de nombreux algorithmes plus ou moins sophistiqués ont été développés pour utiliser cette technique pour toutes sortes d'applications.

Cependant, nous n'avons pas besoin d'une méthode sophistiquée puisque, en particulier, la taille de notre graphe de compatibilité est généralement relativement petite [§ 12.3]. Un algorithme simple et direct, tel que celui décrit en [Bol 83], a prouvé son efficacité sur des applications variées, comme par exemple la stéréovision [Hor 89]. Il représente ainsi un bon compromis.

L'algorithme qui extrait toutes les cliques maximales inclues dans un graphe peut être décrit comme une fonction récursive élémentaire [ALG. 21]. Dans cet algorithme :

- $Candidats$ représente l'ensemble de nœuds qui peut être ajouté à la clique courante, c'est-à-dire relié grâce à un arc à chaque nœud de la clique courante,
- $Voisins(n)$ est l'ensemble des nœuds adjacents au nœud n.

L'efficacité de cet algorithme peut être améliorée en choisissant avec soin le nœud n dans P. Un des critères consiste à sélectionner le nœud qui a le plus petit nombre de voisins. Un taux de confiance

ALG. 21 – MaxClique(*Clique*, *Candidats*)

si $Candidats = \emptyset$ **alors**
  Clique est une clique maximale
sinon
  $P \leftarrow Candidats$
  tant que $P \neq \emptyset$ **faire**
    choisir un nœud $n \in P$
    $P \leftarrow P - \{n\} - \{Voisins(n)\}$
    MaxClique($Clique \cup \{n\}, Voisins(n)$)
  fin tant que
fin si

est associé à chaque clique, défini comme la somme des taux de confiance Δ de chacun des éléments de la clique. Il est utilisé lorsque plusieurs cliques maximales de même taille sont extraites : la clique présentant le taux le plus bas est alors supposée représenter le meilleur appariement.

La clique résultante est finalement utilisée pour calculer la transformation qui permet de recaler $\mathcal{N}_1$ sur $\mathcal{N}_2$. Un appariement i_1^a:i_2^j relie l'indice i_1^a, dont le point caractéristique est (x_1^a,y_1^a), à l'indice i_2^j dont le point caractéristique est (x_2^j,y_2^j). Les coordonnées de ce dernier point peuvent s'exprimer en fonction des coordonnées du premier point de la façon suivante :

$$\begin{aligned} x_2^j &= \overbrace{S\cos\theta}^{kc} x_1^a - S\sin\theta y_1^a + T_x \\ y_2^j &= \underbrace{S\sin\theta}_{ks} x_1^a + S\cos\theta y_1^a + T_y \end{aligned}$$

où T représente la translation, δ l'angle de la rotation et S le changement d'échelle. Les coordonnées des n points du niveau $\mathcal{N}_2$ sont données par le système :

$$\underbrace{\begin{bmatrix} x_2^1 \\ y_2^1 \\ x_2^2 \\ y_2^2 \\ \vdots \\ x_2^n \\ y_2^n \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} x_1^1 & -y_1^1 & 1 & 0 \\ y_1^1 & x_1^1 & 0 & 1 \\ x_1^2 & -y_1^2 & 1 & 0 \\ y_1^2 & x_1^2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^n & -y_1^n & 1 & 0 \\ y_1^n & x_1^n & 0 & 1 \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} kc \\ ks \\ T_x \\ T_y \end{bmatrix}}_{\phi}$$

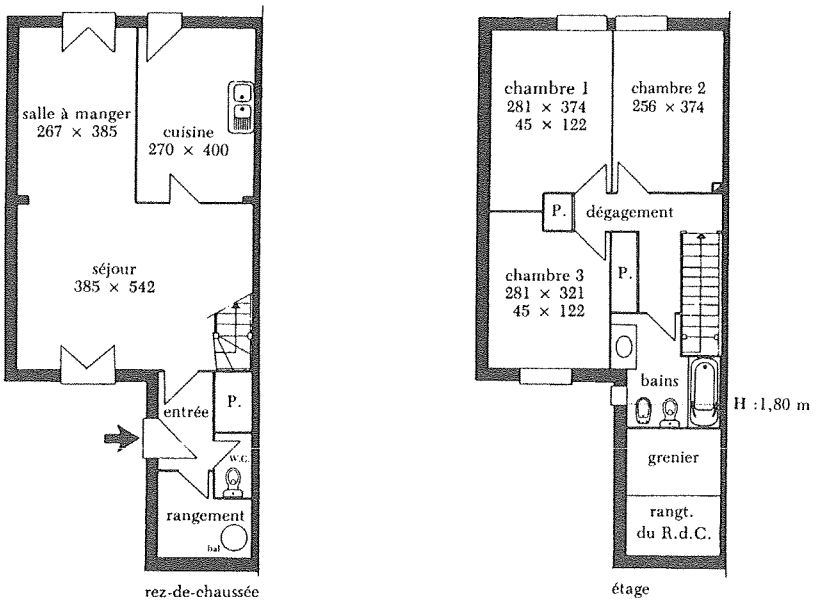
Ce système correspond à la minimisation de l'erreur au sens des moindres carrés. La précision de la solution augmente avec le nombre de points appariés disponibles. La matrice ϕ peut être calculée de la manière suivante [Bal 82] :

$$\begin{aligned} B^tA &= (B^tB)\phi \\ \phi &= (B^tB)^{-1}B^tA \end{aligned}$$

Les facteurs de translation selon les axes X et Y sont respectivement T_x et T_y. L'angle de rotation δ est donné par $\arctan\frac{ks}{kc}$ et le changement d'échelle par $\sqrt{kc^2+ks^2}$.

12.5 Résultats et perspectives

La figure [FIG. 12.3] présente les plans d'un pavillon. Les résultats expérimentaux obtenus à partir de ce pavillon sont présentés [FIG. 12.4]. Le résultat ne semble pas très précis visuellement : il y a en effet

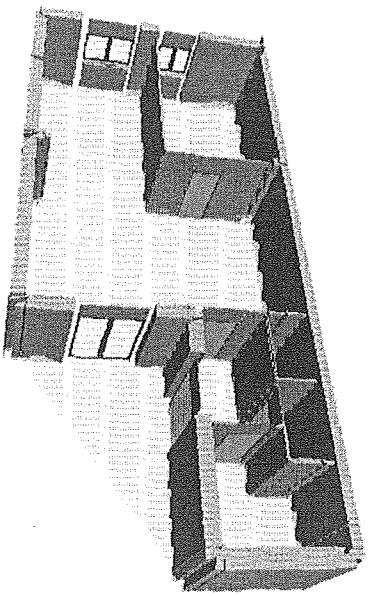


(a) Rez-de-chaussée. (b) Premier étage.

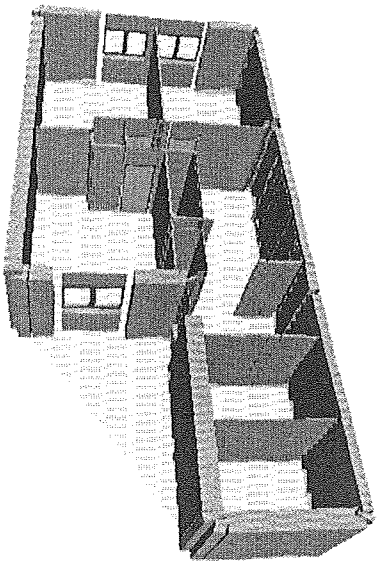
FIG. 12.3 – *Plans des différents niveaux d'un pavillon.*

un petit décalage entre le niveau inférieur et le niveau supérieur. L'étape de mise en correspondance et le recalage calculé à partir de cette mise en correspondance ne sont en fait pas en cause. Ce bruit est issu en fait des artefacts introduits par l'étape de vectorisation. C'est particulièrement le cas sur les coins : la vectorisation génère souvent deux points au voisinage des coins présents sur le plan original au lieu de l'unique point représentant le coin. Par conséquent, deux facteurs biaisent les résultats : la non-unicité de la réponse, et le décalage de la position des coins détectés.

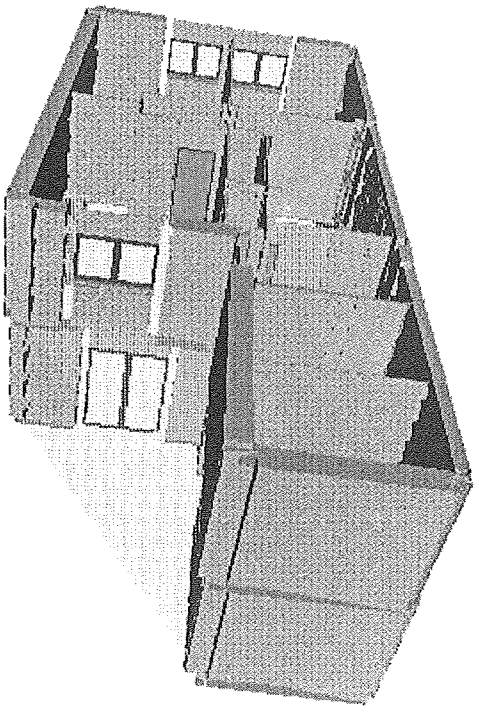
Le traitement est assez rapide à l'exécution : 0.3 seconde pour l'exemple présenté, contenant 79 symboles architecturaux au rez-de-chaussée et 125 au premier étage, d'où ont été extraits respectivement 53 et 51 indices. Le graphe de compatibilité correspondant contient 15 nœuds et 36 arcs, et la plus grande clique maximale de ce graphe contient 5 nœuds.



(a) Structure 3D du rez-de-chaussée.



(b) Structure 3D du premier étage.



(c) Structure 3D du pavillon complet.

FIG. 12.4 – *Reconstruction 3D du pavillon de la figure 12.3 après recalage des structures 3D de ses niveaux.*

Cinquième partie

Mise en œuvre d'un environnement d'analyse

Chapitre 13

Coopération homme-machine et architecture logicielle

13.1 Contexte

Dans les parties précédentes de ce mémoire, nous avons décrit les méthodes que nous avons choisies, adaptées et développées, ainsi que certains des algorithmes qui leur correspondent. Dans cette partie, nous abordons les aspects qui sont essentiellement liés à la mise en œuvre de ces méthodes au sein d'un système informatique. Cette dernière phase recouvre plusieurs étapes, comme le choix d'une architecture logicielle et d'un environnement de développement, le codage des différentes méthodes, leur cadre d'utilisation... L'objectif de ce premier chapitre est de présenter les techniques permettant de prendre en considération ces différents facteurs, en vue de réaliser un système informatique de qualité.

La mise en œuvre d'applications ou de systèmes informatiques conduit de plus en plus les chercheurs à prendre en considération d'autres aspects que ceux purement relatifs à leur domaine. Outre les aspects techniques liés à la programmation, d'autres besoins s'expriment, comme l'évaluation de performances, le choix de protocoles de communication et d'accès aux données, l'ergonomie, la coopération avec l'utilisateur... Nous nous sommes nous aussi intéressé à ces aspects lors de l'élaboration de notre système, et plusieurs points ont retenus notre attention :

- Un nombre conséquent de traitements est nécessaire pour mener à bien notre analyse de plans architecturaux. Afin d'intégrer au mieux les différents traitements de ce système, il est nécessaire de mettre en place un véritable environnement pour les coordonner et les organiser. L'élaboration de tels environnements est une problématique commune à d'autres chercheurs en analyse de documents, ou plus généralement en traitement d'images. Une des initiatives les plus célèbres est IUE[33] [Mun 92, Koh 94], dont l'objectif est de fournir un environnement complet de traitement

33. *Image Understanding Environment.*

d'images. Plus proche de nos travaux, Dori a proposé un formalisme pour la spécification et la représentation des objets et traitements intervenant dans l'analyse de documents techniques. Cet environnement, MDUS[34] [Dor 95a], est assez représentatif de nos besoins sur plusieurs points.

- La diversité de représentation des plans architecturaux ne nous permet pas de fixer précisément les traitements nécessaires à leur analyse, ni leur ordonnancement. D'autre part, certains des traitements que nous avons conçus peuvent s'appliquer sur d'autres types de documents techniques que les plans architecturaux. En effet, l'objectif est de pouvoir réutiliser ces composants d'une application à l'autre. C'est d'ailleurs ce qui nous a poussé à nous doter d'implémentation stable de ces méthodes robustes [Tom 98a].

  Il n'est donc pas possible d'établir une séquence type de traitements à réaliser pour mener à bien une analyse : la séquence à produire est contrainte par le type de représentation utilisée pour décrire le document technique, la qualité de ce document, etc. Par conséquent, il est nécessaire à ce stade d'introduire « l'homme dans la boucle » pour déterminer la séquence de traitements à réaliser pour un document donné.
- Une approche de type « tout automatique », où l'ensemble des traitements nécessaires à l'analyse d'un plan donné seraient lancés séquentiellement, manque de souplesse. Nous devons tenir compte de certaines considérations, comme le bruit contenu dans les données sur lesquelles nous travaillons, ainsi que des limitations de nos méthodes. En effet, chaque étape de l'analyse génère du bruit qui, fournit comme entrée de l'étape suivante, s'accentue progressivement. Afin de fournir en entrée de chacun de nos traitements des données les plus « propres » possibles, nous voulons permettre des interactions sur les données, afin de les corriger ou de les compléter.

Suite à ces constatations, il paraît judicieux de considérer l'implémentation de nos traitements en y incluant des aspects liés à la coopération homme-machine et à l'architecture logicielle. Afin d'intégrer au mieux ces éléments, il convient de présenter d'une part les concepts sous-jacents liés à ces domaines [§ 13.2] et [§ 13.3], et d'autre part une formalisation plus rigoureuse de nos attentes [§ 13.4]. Les choix des modèles retenus sont présentés [§ 13.5]. Pour le détail de l'implémentation de ces choix, le lecteur pourra se reporter aux chapitres 14 et 15 qui présentent la réalisation de notre système.

13.2 Principes de coopération homme-machine

L'utilisation d'ordinateurs et de programmes informatiques est longtemps restée réservée aux seuls informaticiens. En effet, l'austérité du dialogue homme-machine, les problèmes techniques, et la nature des programmes existants cantonnait fortement le cadre d'utilisation de ces nouvelles technologies. Avec la baisse des coûts du matériel informatique et la progression de ses performances, l'informatique s'est progressivement répandue jusqu'à atteindre le succès qu'elle connaît aujourd'hui. Devant l'émergence d'utilisateurs de tous horizons, l'informatique s'est adaptée à ce nouveau public. La recherche de la performance de l'utilisateur face à la machine est très vite devenue un domaine de recherche, exploré par les informaticiens, mais également par d'autres corps de métiers : des psychologues, des graphistes, des éducateurs, des ergonomistes... Les retombées de ces activités de recherche sont intégrables par les concepteurs de logiciels pour la structuration de leurs systèmes informatiques et l'organisation graphique de leurs données [Shn 98].

Afin de mieux comprendre les enjeux et les techniques liés au dialogue homme-machine, nous présentons ici quelques uns de ses principes clés, tant au niveau de la définition même de ce dialogue, qu'au niveau des modélisations d'interaction ou de conception qui sont induites. Cette présentation n'a pas pour

34. *Machine Drawing Understanding System.*

but d'être exhaustive — ni même d'être représentative des derniers travaux réalisés dans ce domaine — mais d'introduire les concepts sous-jacents à notre travail. Le lecteur pourra se référer à [Cou 90, Kol 97] pour un état de l'art de ce domaine.

13.2.1 Introduction

L'interaction homme-machine est un domaine qui tient des sciences cognitives et des techniques informatiques. Une des définitions, proposée par Coutaz [Cou 90], est « *l'ensemble des phénomènes physiques et cognitifs qui interviennent dans la réalisation des tâches informatisées* ». On peut donc considérer que deux aspects sont à prendre en compte pour l'appréhender : l'un dirigé par la psychologie de la communication et l'autre dirigé par les composantes techniques de la réalisation. Historiquement, ces deux aspects ont d'abord évolué en parallèle, et n'ont été rapprochés que par la suite.

La réalisation d'une *interface*[35] suppose tout d'abord la connaissance précise du comportement des entités en présence. Parmi ces entités, on peut considérer deux grandes familles :

- les objets artificiels utilisés en informatique, dont le comportement est généralement bien défini et maîtrisé,
- les sujets humains, pour lesquels on trouve souvent des définitions empiriques, par manque de modèles psychologiques bien adaptés à l'interaction homme-ordinateur.

Définir un dialogue entre ces deux entités nécessite ensuite de déterminer leurs rôles respectifs. Dans ce cadre, il est nécessaire que l'ordinateur ne soit pas perçu comme un simple outil, sans pouvoir décisionnel. Il doit constituer un collaborateur qui participe activement et efficacement à la réalisation d'une tâche. Cette collaboration implique que l'ordinateur doit soutenir l'utilisateur lorsque celui-ci a besoin d'assistance, et que l'utilisateur doit orienter correctement le système. Si la coopération échoue, le dialogue est alors mis en situation d'échec. Afin d'éviter cet échec et de gérer au mieux le dialogue, l'ordinateur doit donc avoir connaissance des facultés et des mécanismes de travail du partenaire.

13.2.2 Modélisation du comportement humain

La modélisation du comportement humain ne s'est pas répandue immédiatement. En effet, cette modélisation reste une tâche compliquée et délicate, ce qui ne facilite pas le travail de l'informaticien. De ce fait, la conception de nombreux systèmes interactifs a souvent été guidée par les exigences informatiques avant de répondre aux réels besoins des utilisateurs. Afin d'intéresser les utilisateurs, cette carence a parfois été compensée par une présentation « tape à l'œil ». Cette approche a cependant rapidement montré ses limites : si une interface n'offre pas la possibilité de compléter les facultés de l'utilisateur, elle ne peut pas faire illusion de façon durable. Il paraît donc indispensable que le concepteur d'un système soit sensibilisé à l'ouverture des logiciels aux concepts de l'ergonomie cognitive et soit guidé dans sa démarche par les connaissances du domaine de l'interaction homme-machine, représentées par des modèles et des principes pratiques. Nous présentons certains de ces modèles qui, s'ils ne sont pas directement utilisés, ont permis d'établir les bases des modélisations actuelles.

35. Le sens associé ici au mot « interface » est relatif à l'ensemble du dialogue homme-machine. Il couvre différents aspects de ce dialogue présentés par la suite. Il n'est pas à confondre avec l'*interface graphique*, qui ne constitue que la *représentation* physique de l'interfaçage.

Modèle du processeur humain

Card *et al.* proposent directement une modélisation de l'être humain : c'est le *modèle du processeur humain* [Car 83]. Ils décomposent celui-ci en trois sous-systèmes (sensoriel, moteur et cognitif) qui sont chacun dotés de mémoire (caractérisée par une capacité, une persistance, du type d'information mémorisée) et d'un processeur. Une des hypothèses retenues dans ce modèle théorique est la capacité d'adaptation de l'utilisateur, dont le comportement est conditionné par l'environnement qu'il utilise. Le modèle proposé est assez simple et présente l'avantage de formaliser des connaissances psychologiques en utilisant une terminologie informatique. Il constitue à ce titre un modèle de base qui a ensuite été complété par d'autres modèles. Tel quel, il n'apporte cependant aucune indication sur la façon d'organiser la conception d'un système interactif.

GOMS

Card *et al.* ont ensuite proposé un modèle de description du comportement, GOMS[36]. Ce modèle, qui repose sur le formalisme du processeur humain, s'intéresse à l'analyse de la tâche à effectuer et de l'évaluation prédictive de la performance liée à l'accomplissement de cette tâche. Pour cela, le comportement est modélisé à plusieurs niveaux d'abstraction :

- les *buts* : structures symboliques qui définissent un état recherché, organisés éventuellement de façon hiérarchique. Un but complexe est alors atteint lorsque tous les sous-buts qui le décomposent sont atteints ;
- les *opérateurs* : actions élémentaires qui provoque un changement d'état (de l'environnement ou de l'opérateur). Intuitivement, ils correspondent aux instructions exécutées par une machine ;
- les *méthodes* : décrivent le procédé permettant d'atteindre un but. Elles représentent le savoir-faire du domaine, et correspondent à la définition donnée au paragraphe [§ 2.2]. Dans le contexte particulier de GOMS, elles correspondent à une suite conditionnelle de buts et d'opérateurs faisant référence au contexte ;
- les *règles de sélection* : permettent d'exprimer le choix d'une méthode lorsque plusieurs méthodes mènent au même but, en fonction d'éléments contextuels.

GOMS apporte un début de solution sur la conception d'un système, approche compatible avec celle des informaticiens : une tâche peut être affinée selon une approche descendante, ou construite selon une approche ascendante. Le modèle est également basé sur l'évaluation de performances : le temps d'exécution de chaque opérateur peut être quantifié, et il est ainsi possible de prédire le temps nécessaire à un utilisateur pour effectuer une tâche. Le modèle reste cependant limité : il n'apporte aucun élément quant à la structuration de la tâche modélisée. Il n'est ainsi qu'un modèle théorique prédictif et quantitatif de performance, qui n'inclut pas la notion d'erreur : il considère un opérateur « idéal ».

36. *Goals, Operators, Methods, Selection.*

Les modèles conceptuels

Les modèles présentés ci-dessus sont intéressants pour comprendre les mécanismes généraux de l'interaction homme-machine. Afin de cerner plus finement la modélisation du comportement, il est utile d'introduire à ce stade deux types de modèles différents :

- le *modèle de conception*, qui est lié à l'outil. Il doit résulter de l'étude des besoins, des possibilités et des limitations de l'utilisateur,
- le *modèle de l'utilisateur*, qui correspond à la représentation que l'utilisateur se fait de l'outil, élaborée à partir de l'image de l'outil. Cette image correspond à la présentation physique, c'est-à-dire à l'interface d'utilisation de l'outil.

En considérant ces modèles, le concepteur a pour objectif de définir une image qui conduise l'utilisateur à construire, au cours de l'interaction avec l'outil, un modèle compatible avec le modèle de conception. Cette approche complète les approches précédentes en y intégrant la notion d'*état du système*, calculé à partir des variables physiques du système. On distingue plusieurs états :

- l'*état effectif*, qui caractérise le modèle de conception,
- l'*état perçu*, qui correspond à la traduction de l'état effectif en fonction du modèle de l'utilisateur, et peut ainsi être décrit sous forme de variables psychologiques.

Les différences entre l'état effectif et l'état perçu peuvent provoquer des erreurs dans le dialogue homme-machine. Il est donc essentiel de limiter ces différences, quoique la correspondance entre les variables physiques et psychologiques est parfois complexe à mettre en œuvre. Cette approche permet ainsi de considérer les sources d'erreurs potentielles, et d'expliquer formellement les difficultés rencontrées par l'utilisateur. Notons qu'il est également possible de considérer un troisième modèle, celui qu'un programme peut élaborer pour représenter l'utilisateur. Ce modèle évolue alors dynamiquement en fonction des caractéristiques de l'utilisateur, et peut ainsi s'adapter à lui.

Si l'état perçu est en adéquation avec l'état effectif, l'utilisateur peut alors plus facilement atteindre le modèle de conception par l'intermédiaire de *tâches*, qui peuvent être décomposées en sept étapes :

1. l'établissement d'un but : c'est-à-dire une représentation mentale de l'état à atteindre. Ce but initial peut être décomposé en une hiérarchie de sous-buts simples,
2. la formation d'une intention, qui résulte de la différence entre le but et la représentation mentale de l'état actuel du système,
3. la spécification d'une suite d'actions, qui correspond à la représentation mentale du plan de résolution,
4. l'exécution des actions, c'est-à-dire la manipulation des dispositifs physiques de commande pour modifier les variables physiques,
5. la perception de l'état du système, représentation mentale de la nouvelle image du système,
6. l'interprétation de l'état du système, en fonction des variables psychologiques impliquées dans le but,
7. l'évaluation de l'état du système par rapport au but.

Cette décomposition permet de structurer le dialogue, en identifiant les points critiques à considérer. Même si elle n'est pas rigoureusement applicable dans tous les types de dialogue, elle fournit au moins une base de formalisation du dialogue.

13.2.3 Principes ergonomiques

Les principes présentés dans le paragraphe précédent sont relatifs à la modélisation théorique du dialogue homme-machine. En abordant les principes ergonomiques, il est maintenant question d'approche expérimentale, qui se veut complémentaire de la première. Les deux approches ont en effet le même objectif, c'est-à-dire l'amélioration de la qualité des interfaces. Les règles liées à l'ergonomie sont nombreuses et évoluent très vite, ce qui rend compliqué leur assimilation. Cela ne facilite pas le travail des informaticiens, qui doivent se mettre régulièrement à jour. On peut toutefois dégager quelques règles fondamentales :

- La *cohérence* : cela recouvre des aspects d'uniformisation de présentation, de précision sur la terminologie utilisée. Le but est le même à chaque fois : faciliter le dialogue avec l'utilisateur en lui permettant de se repérer facilement dans l'environnement. On peut noter que l'utilisation de bibliothèques d'interfaces graphiques permet d'atteindre plus facilement cette cohérence.
- La *concision* : notion qui recouvre deux aspects différents. Le premier est relatif à la clarté de l'information présentée, en évitant de surcharger l'information. Le second correspond au souci de limiter les actions à effectuer par l'opérateur : éviter par exemple qu'une action atomique doive être réalisée par plusieurs interactions. Les utilisateurs de systèmes informatiques introduisent très souvent des actions de ce type, dont les exemples les plus fréquents sont : les abréviations, les raccourcis, les macro-commandes, les opérations de couper–copier–coller, les valeurs par défaut, les actions refaire et défaire...
- Le *retour d'information* : c'est-à-dire la réactivité du système aux interactions de l'utilisateur. Ce retour a pour responsabilité de refléter l'état du système et doit être immédiat et informatif. Il y a au moins trois objectifs à ce retour d'information :
  - rassurer l'utilisateur, par exemple par la présentation d'un sablier ou d'une barre de progression lors de l'exécution de traitements longs, par la présentation de boîtes d'information,
  - réduire la charge cognitive de l'utilisateur, par exemple en lui rappelant le contexte de travail, en lui proposant des barres de défilement pour la navigation, par la présentation grisée des options inaccessibles d'un menu,
  - la détection des erreurs et la proposition de remèdes appropriés dès que possible, par exemple lors de problèmes liés à l'ouverture de fichiers, à la fin anormale de processus.
- La *structuration des activités*, c'est-à-dire l'organisation de l'espace de travail de l'utilisateur en fonction de ses compétences. Une des propositions faites dans ce sens est par exemple de déverrouiller certaines fonctionnalités d'un système au fur et à mesure de sa compréhension par l'utilisateur. Cela conduit à une hiérarchie d'environnements, du plus simple au plus complexe et sous-entend une modélisation dynamique de l'utilisateur tel qu'expliqué [§ 13.2.2].
- La *flexibilité*, c'est-à-dire l'adaptabilité des interfaces. Cette flexibilité repose fortement sur une définition externe des données, utilisée par exemple pour les messages affichés, la définition des raccourcis, les valeurs par défaut, la taille des fenêtres, etc. L'objectif est de pouvoir personnaliser ces paramètres sans avoir à modifier le programme existant. C'est typiquement le cas des fichiers de ressources sous Unix.

Ces règles fournissent une base communément admise en ergonomie, et peuvent être étendues en fonction des spécificités du système à concevoir. On peut notamment y intégrer des notions de confiance, de sécurité, d'intégrité des données, de portabilité... Afin d'évaluer un système interactif, plusieurs facteurs peuvent finalement être considérés, comme le temps d'apprentissage, la rapidité des performances, le taux d'erreur commis par les utilisateurs, la facilité de mémorisation, la satisfaction subjective... Le

lecteur intéressé par un développement de ces principes pourra se référer aux livres présentant les principes d'ergonomie dans la conception d'interfaces homme-machine, comme [Shn 98].

13.2.4 Notes sur la conception

La présentation de principes de coopération homme-machine fait ressortir un domaine vaste, qui dépasse le cadre de compétence des seuls informaticiens, mais nécessite une concertation avec d'autres acteurs, comme des ergonomes. Sans répondre totalement aux problèmes de conception, cette introduction a présenté certains modèles de dialogue, utilisés comme base dans les systèmes de conception actuels. La plupart de ces principes sont intégrés dans les outils disponibles pour la construction d'interfaces interactives [§ 13.3.4]. Cependant, même si ces outils offrent des facilités de conception, il est important de connaître et de garder à l'esprit les principes énoncés ci-dessus lors de l'étape de conception, pour éviter de mauvais emplois de ces outils[37].

13.3 Principes d'architecture logicielle

13.3.1 Introduction

Nous présentons ici quelques principes d'architecture logicielle. Dans notre contexte, ces principes sont influencés par deux domaines de recherche différents. Le premier, le plus naturel, est le génie logiciel qui s'intéresse aux techniques à mettre en œuvre pour produire du logiciel de qualité. La deuxième est la coopération homme-machine évoquée ci-dessus, qui contraint ou conseille l'organisation du système pour favoriser le dialogue homme-machine. Bien entendu, ces deux aspects sont complémentaires, et la coopération homme-machine intègre déjà des paradigmes de génie logiciel pour ses propres besoins, ce qui facilite notre démarche.

De nombreux éléments sont à prendre en considération pour déterminer l'architecture logicielle d'un système informatique. Il existe beaucoup de différences entre les environnements (matériels, outils), le type d'application, les utilisateurs... Un principe de base de génie logiciel émerge cependant : la modularité, principe notamment bien développé grâce aux langages objets. La modularité est utilisée à plusieurs niveaux de granularité : au niveau des applications pour leur conception et également au niveau des systèmes informatiques pour leur organisation.

On peut ainsi décomposer un système informatique logiciel en plusieurs couches hiérarchiquement organisées, appelées *composants fonctionnels*. La couche de plus bas niveau correspond au système d'exploitation de la machine. Viennent ensuite la couche graphique du système d'exploitation, le gestionnaire de fenêtre, l'interface interactive et enfin la gestion du dialogue avec l'utilisateur (sur certaines plateformes, certaines de ces couches sont parfois confondues). C'est à partir de cette organisation que nous devons définir l'architecture de notre système d'analyse de plans. Au niveau terminologique, nous reprenons les définitions introduites par Coutaz [Cou 90]. Nous appelons :

- *applications* : l'ensemble des logiciels relatifs à l'analyse de documents,
- *interface interactive* : le logiciel définissant l'image du système et assurant ainsi la communication entre l'utilisateur et les applications,
- *système interactif* : l'ensemble des applications reliées à une interface.

37. À titre anecdotique, le lecteur pourra visiter le site `http://www.iarchitect.com/` où sont présentés des exemples de bonnes et de mauvaises utilisations d'outils de construction d'interfaces interactives.

13.3.2 Différentes activités dans le développement

Le développement d'un système informatique inclut différentes phases, qui ont été formalisées par le génie logiciel. Celui-ci est né du besoin de concevoir des logiciels de qualité respectant les contraintes de leur cahier des charges [Gau 96]. Il regroupe les techniques théoriques et pratiques permettant de concevoir, réaliser et faire évoluer les logiciels. Ces techniques sont regroupées par *activités*, qui suivent la vie du logiciel :

- l'*analyse des besoins* : c'est-à-dire l'inventaire des besoins par rapport au domaine d'application du logiciel. Cette activité est généralement effectuée en concertation avec les experts du domaine, parfois par des non-informaticiens, l'activité relevant plus des sciences cognitives que des techniques informatiques,
- la *spécification globale* : qui correspond à une première description du système à concevoir par des informaticiens, indépendamment de tout choix technique,
- la *conception architecturale* : qui engage un certain nombre de choix techniques selon le style d'architecture logicielle, comme la décomposition du système à produire en composantes, les communications possibles entre ces composantes par la définition d'interfaces, la représentation des données, les algorithmes,
- la *programmation* : c'est-à-dire le codage de la conception selon un ou plusieurs langages de programmation,
- la *gestion des configurations et intégration* : qui est relative à l'intégration des différentes composantes du système informatique sur la plateforme choisie, et sur sa personnalisation (notamment par des ressources externes),
- la *validation et vérification* : dernières étapes établissant la correction du système produit par rapport aux spécifications énoncées.

Afin d'organiser ces activités, plusieurs modèles de conception ont été proposés : modèle cascade, modèle en V, modèle en spirale, par incrément... Ces différents modèles établissent des cadres d'interactions possibles entre les différentes activités et leur organisation. Ces modèles sont cependant plus adaptés à la conception de systèmes interactifs conséquents qu'à l'interaction homme-machine [Kol 97].

Dans ce schéma, les deux premières activités ainsi qu'une partie de la troisième ont été décrites dans les chapitre précédents, en accord avec les spécificités de notre domaine de recherche. Nous détaillons ci-dessous les options possibles pour l'architecture logicielle afin d'intégrer toutes les applications présentées et le dialogue homme-machine.

13.3.3 Modèles d'architectures de système interactifs

Il existe de nombreux styles d'architecture logicielle, définis pour faciliter la conception des systèmes informatiques, dont la taille et la complexité ont significativement augmentées. Ces styles peuvent se combiner au sein d'un même système informatique : le système est décomposé en plusieurs parties, chacune organisée selon un style. Parmi ces styles, on peut citer le *batch séquentiel* ou les *filtres et tuyaux* qui sont particulièrement bien adaptés aux traitements modulaires exécutés en cascade, mais qui sont peu adaptés à l'interactivité. Leur principe est celui d'un fonctionnement de type *boîte noire*, où un flot d'entrée (les données) est transformé incrémentalement en flot de sortie (les résultats). Ce style d'architecture est celui que nous avons choisi pour l'enchaînement des applications de traitement d'images, avec une communication de données effectuée *via* des fichiers. Pour une bonne introduction à l'architecture logicielle et aux différents styles d'architecture, le lecteur pourra se référer à [Gar 00].

Historiquement, les modèles d'architectures de systèmes interactifs ne sont intervenus que très récemment avec la définition en 1984 du modèle *Seeheim*. Celui-ci repose sur la définition d'un langage pour formaliser le dialogue homme-machine. Comme dans tout langage, on peut distinguer trois niveaux dans ce langage : lexical, syntaxique et sémantique. Le modèle *Seeheim* est le premier à avoir formalisé ce type de langage sous une forme générique, et comprend trois composantes :

- l'*interface avec l'application* : son rôle est de convertir les informations entre les applications et le contrôleur du dialogue,
- le *contrôleur du dialogue* : oriente le dialogue homme-machine à partir des informations reçues de l'interface avec l'application et de la présentation. Il synthétise ces informations, vérifie leur correction à partir du langage défini, et envoie des requêtes aux deux autres composantes,
- la *présentation* : détermine l'image du système présentée à l'utilisateur et permet à l'utilisateur d'interagir. Les interactions sont ensuite dirigées vers le contrôleur du dialogue.

Ce modèle présente l'avantage d'être compatible avec les modèles présentés en génie logiciel, notamment au niveau de la modularité. En effet, dans cette organisation, il est possible de changer chaque composante sans modifier les autres. Ce type d'approche permet de définir un cadre générique de dialogue et d'architecture logicielle.

Le modèle *entrée/sortie* met l'accent sur l'échange et la transformation des informations. Dans ce modèle, le système informatique est décomposé en couches hiérarchiques selon le même principe que les composants fonctionnels. La particularité de ce système réside dans le fait que chaque couche peut être atteinte sans passer automatiquement par la précédente. Les communications entre composants reposent sur un schéma type client-serveur. Le modèle *entrée/sortie* préserve donc une certaine flexibilité d'utilisation tout en apportant une structuration. Il ne fournit cependant que de grandes lignes sur la structuration des traitements, dont une partie est accessible maintenant en standard sous tout environnement.

Enfin, le modèle *multiagent* s'inspire des systèmes de type stimuli-réponse. Le modèle décrit un système informatique comme un ensemble d'agents dotés de réactivité. Chaque agent peut traiter des stimuli et peut produire à son tour des stimuli, c'est-à-dire des événements. Chaque entité manipulée (événement, agent) appartient à une classe qui en décrit les caractéristiques. L'organisation résultante est très modulaire. Le modèle propose une exécution en parallèle des traitements et une communication par événement. Le modèle *à objet* est issu de ce modèle *multiagent*. Il propose de décrire tout un système informatique en termes de classes. Une classe, dotée de propriétés (attributs, opérateurs), sert de matrice pour créer des instances de classes aux propriétés identiques. Des contraintes d'utilisation ou d'état peuvent être associées aux classes afin d'en contrôler le comportement. Il est également possible de mettre en pratique des règles de masquage de l'information et d'héritage. Toutes ces fonctionnalités permettent de concevoir des systèmes réutilisables, flexibles, modulaires et extensibles. Pour une description plus complète des méthodes à objets et de la conception sous-jacente, le lecteur pourra se référer à [Mey 91].

Tous les modèles présentés permettent de dégager certaines règles complémentaires d'organisation de systèmes informatiques interactifs. Afin de rassembler les avantages de différents modèles, des modèles plus élaborés ont été conçus. C'est le cas du modèle PAC qui structure l'architecture d'un système récursivement, en distinguant les fonctionnalités du système de leur présentation. Il introduit de plus une notion de contrôle : le contrôleur gère les ressources (processeur, synchronisation, écran) et sert d'interface entre la présentation et son abstraction. De nombreuses applications découlent de ce modèle, très utile quand beaucoup de paramètres sont à prendre en compte à un instant donné. C'est également le cas du modèle MVC[38] introduit en Smalltalk, qui définit trois objets : le *modèle* qui réunit l'ensemble des

38. *Model, View, Controller.*

fonctions abstraites, la *vue* qui définit la présentation de sortie et le *contrôleur* qui interprète les entrées provenant de l'utilisateur. MVC ne présente cependant pas le même caractère générique que PAC et reste très lié au langage Smalltalk. Pour une comparaison des avantages et inconvénients de ces deux modèles, le lecteur pourra se reporter à [Cou 90].

Plus récemment, le modèle ∇ a été proposé pour compenser les carences du génie logiciel face aux systèmes interactifs [Kol 97]. Pour cela, le modèle ∇ intègre le facteur humain dans la conception du système et propose des concepts permettant de favoriser la réutilisabilité des composantes du dialogue homme-machine.

13.3.4 Outils pour la construction d'interfaces

Outre l'organisation du système informatique, les choix d'architecture logicielle concernent aussi ceux de la mise en œuvre des interfaces interactives. Les différents modèles d'architecture présentés ont été utilisés pour définir des outils logiciels, réalisant les abstractions décrites. Il en existe trois grandes familles :

- Les *boîtes à outils*, bibliothèques de procédures qui proposent des composantes comme les événements, les *widgets*[39]... Ces boîtes à outils offrent certains avantages. Certains sont liés à l'utilisation d'une bibliothèque en général : fiabilité, documentation, efficacité, facilité d'utilisation, portabilité. Celles basées sur un langage à objets permettent de plus de favoriser la modularité et la réutilisabilité du système produit. Les boîtes à outils présentent plus spécifiquement d'autres avantages inhérents à leur domaine d'application, le dialogue homme-machine. On trouve ainsi une bonne intégration des critères ergonomiques, assurant une cohérence de la présentation, la séparation au niveau des composantes de dialogue et de présentation, parfois la possibilité d'utiliser une représentation externe des ressources et généralement un *protocole embarqué*, c'est-à-dire une gestion des événements intégrée à chaque composante de la boîte à outils. Ce mécanisme permet de faciliter la structuration d'un système interactif, même si en contrepartie il peut restreindre la latitude d'intervention du concepteur. Cette contrepartie tend cependant à disparaître dans des boîtes à outils récentes, comme par exemple QT dont le protocole est basé sur des *signaux* et des *slots* [Dal 99]. Les boîtes à outils constituent un outil précieux pour les informaticiens qui trouvent, plus encore qu'un outil logiciel, un guide permettant de suivre des principes de coopération homme-machine pour le développement de leurs systèmes interactifs.
- Les *squelettes d'application*, qui sont construits à partir de boîtes à outils. Ils correspondent à des patrons d'applications, permettant de mettre en place la structure minimale d'un type d'applications reposant sur une boîte à outils. Les composants spécifiques aux systèmes à développer viennent ensuite se greffer sur ces patrons. Ces squelettes sont cependant délaissés suite à l'essor des boîtes à outils qui, plus faciles d'utilisation, diminuent l'intérêt de cette voie, peu adaptée à la réutilisabilité.
- Les *générateurs d'interfaces* à partir de spécifications. L'objectif de ces générateurs est de simplifier la construction des interfaces interactives, qui n'est pas toujours aisée lorsqu'elle repose uniquement sur les services offerts par la boîte à outils. Plusieurs formalisations de cette approche ont été proposées, à base de diagrammes de transition, de réseaux de Petri, de grammaires hors contexte... Ces formalisations montrent cependant leurs limites, ne permettant généralement que d'exprimer une partie du dialogue homme-machine. Pour assister la conception d'interface interactives, les concepteurs s'orientent plutôt vers la construction interactive des interfaces interactives, grâce à des applications disponibles avec les boîtes à outils.

39. *Window gadget.*

13.4 Étude des qualités d'organisation requises

Pour intégrer nos différentes applications, nous avons décidé de concevoir un environnement complet d'analyse de documents techniques et de plans architecturaux en particulier. Nous souhaitons doter cet environnement de capacités d'interaction avec l'utilisateur, afin de concevoir le processus d'analyse comme une coopération homme-machine. Cet environnement s'accorde également avec la solution logicielle que nous devions fournir au CNET. Le cahier des charges de notre contrat prévoyait certaines contraintes, que nous avons intégrées et complétées. Nous avons ainsi conçu une liste des qualités attendues de notre environnement. Cette liste ne se veut pas exhaustive, mais permet de fixer le cadre de développement et d'orienter nos choix de conception :

- Toutes les applications développées pour l'analyse de documents techniques doivent être intégrées dans notre environnement. Elles doivent pouvoir être exécutées de façon conviviale selon un enchaînement à fixer par l'utilisateur en fonction du type de document à analyser.
- Il faut pouvoir exploiter au mieux les compétences homme-machine. Le dialogue ne doit pas se limiter à une simple encapsulation des traitements par une interface interactive, mais doit correspondre à une véritable coopération homme-machine. Dans ce contexte, l'utilisateur doit être :
  - superviseur de l'analyse, en choisissant les applications à mettre en œuvre et la valeur des paramètres attendus par ces applications. Il doit aussi pouvoir effectuer des retours en arrière si ses choix ne s'avèrent pas pertinents,
  - opérateur, en ayant la possibilité d'interagir sur les données manipulées, pour les compléter, les modifier, les corriger entre deux traitements automatiques. L'objectif est de casser les chaînes de traitements monolithiques : l'utilisateur doit pouvoir interagir entre chaque phase automatique afin d'être en mesure de fournir les données les plus « propres » possibles comme entrée du traitement suivant.

  Nous nous orientons ainsi vers la définition d'un environnement d'*assistance* à l'utilisateur. Le nombre réduit de paramètres associé à chaque traitement et une présentation claire et contextuelle des traitements applicables doit permettre à l'utilisateur d'établir de manière concise les actions qu'il souhaite réaliser.
- L'environnement résultant doit être modulaire, facilement maintenable et extensible. Dans ce cadre, la réutilisabilité doit constituer une préoccupation principale, et l'architecture globale adoptée doit favoriser cette notion. L'exploitation de représentations externes des ressources est dans ce cadre un élément appréciable.
- Les applications d'analyse doivent pouvoir être exécutables en dehors de tout contexte interactif. Cette fonctionnalité est en effet appréciable pour traiter de gros volumes de données, comme un ensemble homogène de plans pour lequel un ensemble de valeurs paramètres est déterminée. Il est souhaitable dans ce cas de ne pas monopoliser l'utilisateur alors que le déroulement de l'analyse est établi.

13.5 Principes utilisés pour la mise en œuvre

Nous avons présenté les concepts de base de la conception en dialogue homme-machine, en architecture logicielle, ainsi que nos attentes d'un environnement d'analyse de documents techniques. Les techniques de coopération homme-machine font ressortir la nécessité de modéliser l'utilisateur, et de le différencier du modèle de conception, lié dans notre cas à l'analyse de documents. Des modèles d'architecture logicielle pour les systèmes interactifs ont été proposés. Ceux-ci montrent cependant leurs

avantages sur des systèmes conséquents ou sur les systèmes où le dialogue homme-machine suit un schéma relativement bien établi et peut donc être modélisé. Nos besoins, s'exprimant selon un schéma semi-séquentiel, sont plus modestes. En effet, nous manipulons en fait trois sortes d'images différentes : des images à niveaux de gris, en noir et blanc, et des images vectorielles. La plupart des traitements que nous appliquons sur ces images sont exécutés sans enchaînement préétabli, et laissés à l'appréciation de l'utilisateur. De même, les interactions que nous souhaitons mettre en œuvre ne sont pas étroitement liées aux traitements appliqués, mais bien plus à la nature des images manipulées.

Nous sommes cependant attentifs aux principes énoncés, et nos choix sont guidés par ces principes [Dos 99a]. Parmi ces choix :

- *Intégration de paradigmes de génie logiciel et utilisation d'un langage à objets.* Nous voulions de plus faire la transition du code précédent (langage C) vers le nouveau langage le plus facilement possible. Nous avons donc choisi le langage C++ qui, s'il n'est pas nécessairement le meilleur langage à objets, est plus ou moins un standard dans l'industrie, et garantit une bonne compatibilité avec le code existant en C. Cette approche est également utilisée pour la mise en œuvre d'algorithmes de traitement d'images par Géraud *et al.* [Gér 99].
- *Choix des outils.* Nous avons décidé de nous doter d'un certain nombre d'outils pour réaliser notre système et nous affranchir des problèmes résolus dans la littérature :
  - c'est le cas de certaines structures de données, comme les containers (listes, tables, ensembles, etc.) et algorithmes associés, où nous avons adopté STL[40] [Mus 96], bibliothèque standard C++ intégrée maintenant dans tout compilateur C++,
  - pour les structures de graphes et certains algorithmes géométriques, nous avons choisi la bibliothèque LEDA[41] [Meh 00, Meh 99], conçue au *Max-Planck-Institut für Informatik* à Sarrebruck, mise à disposition gracieusement pour les activités de recherche. Cette bibliothèque est décrite comme la meilleure mise en œuvre en C++ des structures de graphes par [Ski 98].
  - pour la documentation, appendice indispensable afin de fournir des informations détaillées sur les composantes de notre système, nous avons choisi DOC++ [Wun 00]. Cet outil génère automatiquement une documentation à partir du code C++, dans un format hypertexte (HTML) ou prêt à l'impression (LaTeX). Son principe est très similaire à celui de `javadoc` : l'analyse de commentaires spéciaux insérés dans le code source, ce qui permet d'assurer une cohérence entre les sources des programmes et leur documentation.
- *Formats d'entrée/sortie.* Nous souhaitons utiliser autant que possible des standards ouverts, si possible du domaine public, disponibles facilement. Nous avons arrêté nos choix[42] sur :
  - le format PBM[43] de Jef Poskanzer pour la représentation des images numérisées. Ce format est bien documenté et de nombreux outils de conversion de ou vers ce format sont disponibles. Ils sont maintenant inclus dans la majeure partie des solutions Unix ;
  - le format DXF[44] pour la représentation des images vectorielles 2D ;
  - le format VRML[45] pour la représentation des images vectorielles 3D.
- *Boîte à outils pour la construction d'interfaces interactives.* Une grande gamme de boîte à outils était disponible au moment où la réalisation de cet environnement a commencée, il y a 4 ans. L'in-

40. *Standard Template Library.*
41. *Library of Efficient Data Types and Algorithms.*
42. Le lecteur peut trouver une description des différents formats utilisés à partir du site « *Wotsit's Format* » [Oli 00].
43. *Portable Bitmap.*
44. *Drawing Interchange Format.*
45. *Virtual Reality Modeling Language.*

terface interactive à réaliser n'étant pas un simple *front-end*, mais devant comporter des possibilités d'interaction, nous avons privilégié les boîtes à outils reposant sur C++. Ce type d'outils combine en effet deux avantages : l'utilisation d'un langage à objets pour la construction d'interfaces interactives, et la possibilité d'utiliser un langage complet pour mettre en œuvre les traitements interactifs. Dans le cadre de notre contrat avec le CNET, nous avons choisi le logiciel commercial ILOG VIEWS. Celui-ci comprend une bibliothèque de classes C++, ainsi qu'une application permettant de créer interactivement des interfaces interactives. Les fonctionnalités d'ILOG VIEWS sont détaillées en annexe A.

D'autre part, afin que notre système soit flexible et facilement maintenable, nous avons opté pour une décomposition en trois couches [FIG. 13.1] :

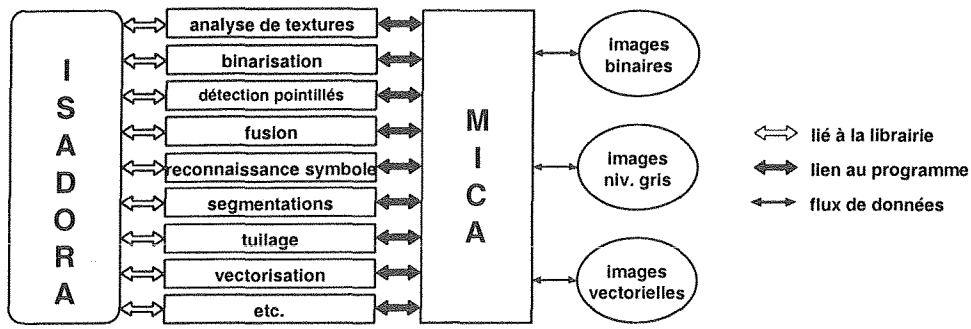


FIG. 13.1 – *L'organisation en différentes couches de notre système.*

- la première couche correspond à une bibliothèque de classes C++, appelée ISADORA, dédiée à la reconnaissance de graphiques. Elle capitalise le savoir-faire générique en analyse de documents techniques développé par l'équipe. Une description de cette bibliothèque et des exemples d'utilisation sont présentés chapitre 14 ;
- la seconde couche est une couche applicative, qui comprend des applications de reconnaissance de graphiques. Dans cette couche, chacun des traitements présenté dans les chapitres précédents correspond à une application. Les applications bas-niveau correspondent plus ou moins à un encapsulage d'appels aux composantes d'ISADORA, tandis que celles de plus haut niveau sont de « réelles » applications, incluant des appels aux fonctionnalités offertes par ISADORA. Toutes ces applications sont des programmes indépendants : elles peuvent être exécutées en ligne de commande et être facilement remplacées lorsqu'un algorithme plus efficace est implémenté. La communication des données entre ces applications est réalisée *via* des fichiers ;
- enfin, la troisième couche est une interface interactive, qui permet de placer « l'homme dans la boucle ». Cette interface encapsule toutes les applications de la seconde couche, qu'elle peut exécuter au moyen d'un simple lien. Elle permet également de visualiser les images qui sont manipulées, d'interagir avec ces données, et apporte une assistance à l'utilisateur en le guidant, en lui proposant des choix par défaut, en personnalisant son environnement de travail en fonction du contexte... Cette troisième couche est présentée plus en détails dans le chapitre 15.

Chapitre 14

ISADORA : une bibliothèque de reconnaissance de graphiques

14.1 Organisation générale de la librairie

La bibliothèque ISADORA correspond à la première couche du système que nous avons défini [§ 13.5]. Elle est constituée d'un ensemble de classes C++ implémentant des méthodes de base de reconnaissance de graphiques. On trouve parmi ces méthodes certains des traitements bas-niveau présentés chapitre 3, certaines composantes de nos méthodes de vectorisation (chapitre 5), ainsi que des méthodes implémentant des opérateurs, tels que les masques de convolution, les calcul de Gradient, de Laplacien...

Le problème fondamental que nous avons dû résoudre est celui de la mise en œuvre d'opérations de traitement d'images grâce à un langage à objets. Le paradigme de la conception objet est basé sur l'encapsulation de données, c'est-à-dire la description des types de données abstraits et de leur interface, alors que le traitement d'images est généralement vu comme une collection d'opérateurs appliqués aux images. Par le biais d'un exemple simple, considérons la convolution d'une image par une Gaussienne, ce qui permet d'obtenir une nouvelle image $J = I \otimes G$. La question est alors : la convolution doit-elle être définie comme une fonction membre de la classe `Image` ou doit-elle être représentée par un opérateur global ? Plusieurs solutions ont été étudiées pour résoudre ce dilemme et nous avons été très attentif au formalisme proposé par IUE [Koh 94], qui est également à la base de la méthodologie objet proposée par Dori [Dor 95a].

Pour rester pragmatiques, nous avons adopté une solution plus proche de celles proposées dans certaines bibliothèques commerciales, comme *ImageVision Library*™ de Silicon Graphics [Gra 00]. L'idée de base est assez simple. La classe `Image` est la classe de base d'une hiérarchie. Ses classes dérivées définissent les différents types d'images : `GreyLevelImage`, `BinaryImage`, `FloatImage`... Pour chacune de ces classes, chaque opération de traitement d'images correspond à une classe dérivée et est implémentée par un constructeur de cette classe dérivée. Différentes méthodes pour effectuer la même opération

conceptuelle peuvent ainsi être facilement implémentées à travers les classes dérivées d'une classe abstraite. Bien sûr, ce paradigme ne s'applique pas seulement au traitement d'images, mais également à toute l'analyse. Cela présente l'avantage de rendre le code lisible : les concepteurs aussi bien que les clients de la bibliothèque écrivent un code compact et clair.

De cette manière, nous espérons nous être doté d'une bibliothèque complète, composée de composantes robustes qui peuvent être utilisées d'une application à une autre.

14.2 Étude de la hiérarchie des classes implémentées

Les classes de la bibliothèque ISADORA sont organisées hiérarchiquement comme un arbre, avec une classe racine appelée `IsaObject`, contenant certaines informations communes à tous les objets, comme certaines fonctionnalités pour la manipulation d'erreurs. Cet arbre peut être vu comme une collection de sous-arbres regroupant chacun des classes correspondant aux différents types d'objets à manipuler pour la conception d'un système d'analyse de documents. Chaque sous-arbre est organisé de la même manière. Il y a trois familles principales de classes :

- *Les classes dédiées au traitement d'images.* Les classes concernées sont principalement les classes dérivées des classes `Image` [FIG. 14.1], `Mask` et `Histogram`. Les images peuvent représenter des données de traitements ou le résultat d'un traitement. Elles sont représentées comme des tableaux de points. Nous avons intentionnellement évité d'utiliser une définition trop générale, pour éviter la complexité trouvée par exemple dans les spécifications de IUE. La plupart des traitements d'images sont disponibles : calcul d'histogramme, traitements à base de convolution, morphologie mathématique, détection de contours, etc.
- *Les classes dédiées au traitement de graphiques* (sous-graphe de racine `Graphics`). Cette hiérarchie inclut toutes les sortes de primitives graphiques, qui résultent des différents modules de segmentation, et également les regroupements de ces primitives, résultant des modules d'analyse : points, segments, chaînes de segments, rectangles, arcs de cercle, composantes connexes, etc. aussi bien que les collections de ces objets.
- *Les classes utilitaires.* C'est le cas par exemple de la hiérarchie dont la racine est la classe `IsaFile`. Ces classes sont utilisées pour le stockage des images numérisées ou des primitives graphiques dans un format donné.

Au total, ISADORA regroupe un peu plus d'une soixantaine de classes représentant plus de 30 000 lignes de C++.

14.3 Exemples d'utilisation

Plusieurs exemples sont présentés. Le premier met l'accent sur le caractère générique de notre implémentation. Le second est un exemple de programme réalisable à partir d'ISADORA : une détection de contours.

14.3.1 Chaînage

Une fois que le squelette d'une image est calculé, les points du squelette doivent être chaînés. Nous utilisons à cette fin la classe `LinkedChainList` qui permet de calculer la représentation d'un squelette

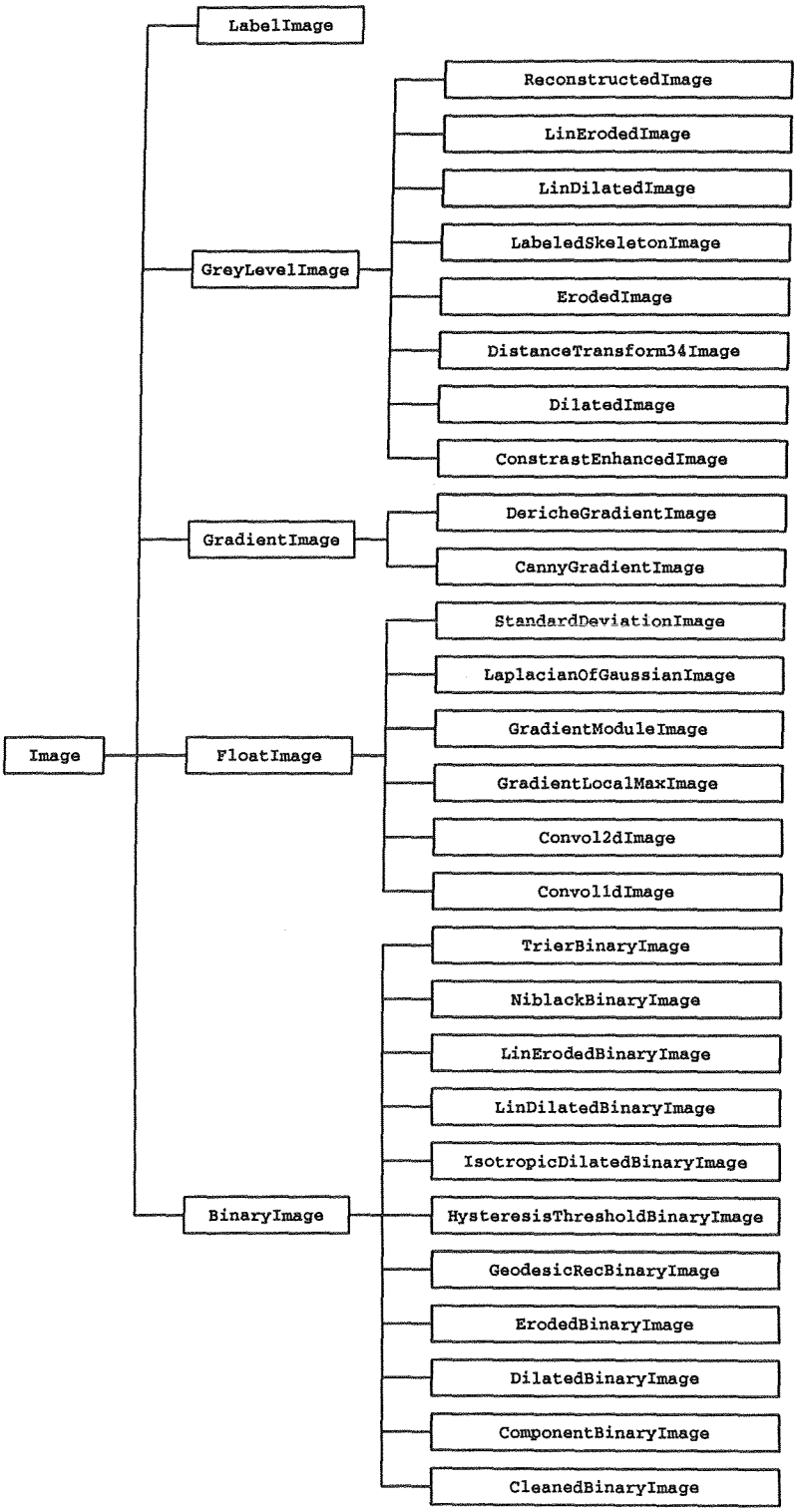


FIG. 14.1 – *L'arborescence de la classe* `Image`*. Les classes immédiatement dérivées de la classe* `Image` *représentent les différents types d'image supportés. De ces classes sont dérivées les différentes opérations de traitement d'images implémentées.*

en une liste de points 2D chaînés [ALG. 6, page 55]. Nous développons ici le contexte de cette classe, pour illustrer comment une mise en œuvre spécifique, comme l'utilisation d'une liste de points pour représenter une chaîne, peut être encapsulé dans une classe générique abstraite.

Premièrement, nous avons défini une classe générique `GenChain` qui fournit une interface générique pour n'importe quel type de chaîne, indépendamment de la façon dont cette chaîne est représentée en interne [PROG. 14.1]. On peut noter l'utilisation de méthodes virtuelles pures qui confèrent à la classe son statut de classe abstraite. Il est ensuite possible de concevoir une mise en œuvre possible de ce type de chaînes, en utilisant une liste de points référencés par un itérateur [PROG. 14.2]. Bien sûr, les méthodes déclarées dans l'interface générique doivent être définies par la suite. Finalement, un chaînage complet peut être défini comme une liste de telles chaînes, avec des coordonnées entières [PROG. 14.3]. Cette

```
1   template <class T>
2   class GenChain : public Graphics {
3   public:
4     /// Constructeur.
5     GenChain() {}
6
7     /// Longueur de la chaîne (nombre de points).
8     virtual int length() const = 0;
9
10    /// Premier point de la chaîne.
11    virtual GenPoint<T> first() const = 0;
12
13    /// Dernier point de la chaîne.
14    virtual GenPoint<T> last() const = 0;
15
16    /// Positionnement de l'itérateur au début de la chaîne.
17    virtual void setInitialPosition() = 0;
18
19    /** Accès au point suivant de celui pointé par l'itérateur.
20        La fonction retourne 'true' s'il est possible d'avancer
21        dans la chaîne, et 'false' sinon.
22        Quand le déplacement est possible, les coordonnées (x,y)
23        du nouveau point sont stockées dans 'transPoint' et la
24        direction du nouveau code est stockée dans 'dir'.
25    */
26    virtual int next(GenPoint<T>& transPoint, Direction& dir) = 0;
27
28    /// Inverser la chaîne.
29    virtual void reverse() = 0;
30
31    /// La chaîne courante est-elle vide ?
32    virtual int empty() const = 0;
33  };
```

PROG. 14.1 – *Extrait de l'interface de la classe* `GenChain`.

dernière classe est dotée d'un constructeur général paramétré par un objet de type `BinaryImage`, qui procède à un suivi récursif des chaînes de points. Nous utilisons ce constructeur pour chaîner les points du contour après une étape de détection de contours. Cependant, afin de pouvoir effectuer un chaînage sur un squelette, nous avons ajouté un deuxième constructeur à la classe, afin de construire une liste de

```
1  template <class T>
2  class GenLinkedChain : public GenChain<T> {
3  protected:
4    /// La liste elle-même.
5    list< GenPoint<T> > theList;
6
7    /// Un itérateur sur la liste.
8    list< GenPoint<T> >::iterator theIter;
9
10 public:
11   // Implémentation de l'interface définie par GenChain<T>
12 };
```

PROG. 14.2 – *Extrait de l'interface de la classe `GenLinkedChain`.*

```
1  class LinkedChainList : public list < GenLinkedChain <int> >
2  {
3    ...
4  };
```

PROG. 14.3 – *Extrait de l'interface de la classe `LinkedChainList`.*

points chaînés à partir d'un objet de type `LabeledSkeleton`. Suivant le type de données à traiter, il ne reste alors plus qu'à appeler le bon constructeur.

14.3.2 Détection de contours

Afin d'illustrer le caractère réutilisable de notre bibliothèque, nous présentons ici un exemple de programme qui n'est pas utilisé dans notre analyse de plans architecturaux, mais qui peut être conçu à partir d'ISADORA. Dans cet exemple [PROG. 14.4], nous montrons comment il est possible de concevoir un programme d'une dizaine de lignes effectuant une détection de contours, et comment notre structure modulaire permet très simplement de remplacer une des étapes de ce traitement par une autre.

14.4 Les traitements de la couche applicative

Comme expliqué [§ 13.5], les traitements de la deuxième couche de notre système, la couche applicative, sont des programmes indépendants qui reposent sur la bibliothèque ISADORA. Les traitements correspondant au bas-niveau de notre système (binarisations, segmentations) sont de simples encapsulages réalisées autour des composantes ISADORA correspondantes. Les traitements de plus haut-niveau sont en revanche de réelles applications qui profitent des fonctionnalités offertes par ISADORA, notamment pour la manipulation des fichiers, des structures de données, etc. Leur principe de mise en œuvre est très proche de celui des exemples d'utilisation d'ISADORA présentés ci-dessus.

```
1   PgmFile f("image.pgm");   // Fichier contenant l'image originale
2   GreyLevelImage myImg(f);  // Le charger comme image à niveaux de gris
3
4   // Calcul de l'opérateur Canny avec sigma == 1.2
5   CannyGradientImage myDeriv(myImg, 1.2);
6
7   // Pour choisir l'opérateur de Deriche plutôt que celui de
8   // Canny, commenter la ligne précédente et décommenter suivante
9
10  // DericheGradientImage myDeriv(myImg);   // Paramètres par défaut
11
12  // Max. de gradient
13  GradientLocalMaxImage maxG(myDeriv);
14
15  // Extraction des contours par seuillage hysteresis
16  HysteresisThresholdBinaryImage myEdge(maxG, 0, 5);
17
18  // Chaînage des contours
19  LinkedChainList myChains(myEdge,1,0);
20
21  // Il est ensuite possible de calculer une approximation
22  // polygonale sur les chaînes, de sauvegarder les segments
23  // résultants dans un fichier DXF, etc.
```

PROG. 14.4 – *Squelette de programme de détection de contours.*

Chapitre 15

Interface et interactions homme-machine

15.1 Introduction

Nous avons mis en évidence dans les chapitres précédents l'importance de la coopération homme-machine dans l'environnement d'analyse que nous développons. En particulier, nous avons montré les limites d'une approche « tout-automatique » et présenté les avantages d'un dialogue afin de superviser le processus d'analyse et d'assister l'utilisateur pour l'interaction avec les données. Cette constatation nous a conduit assez tôt à envisager l'intégration d'une interface homme-machine interactive dans notre système [AS 98c] ainsi qu'une architecture logicielle *ad hoc* pour intégrer cette interface [Dos 99a, Dos 00c]. Rappelons que le développement d'un système complet comme le notre n'est pas une action isolée, mais est également la préoccupation d'autres chercheurs du domaine en analyse de document [Bap 98, Dor 95a]. Nous présentons dans ce chapitre les techniques employées pour mettre en œuvre cette interface interactive, appelée MICA [46], et le dialogue homme-machine.

15.2 Démarche générale adoptée sous MICA

Nous rappelons ici les fonctionnalités principales prises en compte pour l'élaboration de notre interface homme-machine :

- *Assistance de l'utilisateur dans le processus d'analyse.* Afin d'apporter une assistance adaptée aux besoins de l'utilisateur, l'environnement doit s'accorder avec la progression de l'analyse. L'utilisateur doit toutefois garder la possibilité de revenir à chacune des étapes qu'il a réalisée.

46. Modélisation Interactive de Cartes Architecturales.

- *Visualisation des données manipulées lors des différentes phases d'analyse.* Afin de superviser le processus d'analyse, l'utilisateur doit pouvoir suivre de manière permanente l'évolution des résultats engendrés par les traitements qu'il exécute. Cette contrainte implique que tous les types d'image manipulées doivent être supporté par le visualisateur.
- *Interactions pour la correction.* Afin de manipuler les données les plus propres possibles à chaque étape de l'analyse, l'utilisateur doit avoir des possibilités d'interaction sur les données.

Ces considérations fournissent un cadre de réflexion pour la conception de la troisième couche de notre système d'analyse de plans [FIG. 13.1, page 157]. En particulier, nous devons nous interroger sur la nature du dialogue homme-machine. Les principes énoncés au chapitre 13 nous fournissent un guide de développement du système interactif. Il est cependant difficile de l'appliquer à la lettre : notre système contient un certain nombre de particularités — choix de l'enchaînement des traitements dépendant du type de plans, images complexes et bruitées — qui ne le prédispose pas à l'application d'une méthodologie traditionnelle. Cette constatation est d'ailleurs relevée par d'autres chercheurs [Clo 99]. On peut cependant faire les constatations suivantes :

- Trois types d'images peuvent être manipulés au cours de l'analyse : les images numérisées en niveaux de gris, les images numérisées binaires et les images vectorielles 2D. Même si la succession de traitements à appliquer dépend du contenu de chaque image initiale, on peut considérer que :
  - ces trois types d'image correspondent à la progression de l'analyse (sans tenir compte ici des éventuels retours en arrière),
  - des traitements correspondent à ces trois types d'images, ainsi que certaines interactions.

  Il faut de plus considérer un quatrième type d'image : celui des images vectorielles 3D. Les images de ce format correspondent aux résultats finaux que nous générons. Nous ne supportons ainsi pas ce type d'image dans notre environnement : nous utilisons à cette fin un visualisateur VRML, tel que VRWEB [Pic 00].
- Les applications que nous avons conçues (deuxième couche logicielle de notre système) acceptent et génèrent des images dont le type est l'un des trois énuméré ci-avant. Nous devons par ailleurs permettre à l'utilisateur de revenir en arrière à n'importe quel stade de l'analyse. Nous avons vu que le style d'architecture de cette couche se définit comme un *batch séquentiel* [§ 13.3.3], ce qui induit une communication des données entre les différentes applications par fichiers. Dans ce cadre, un fichier est donc associé à chaque image générée par un traitement. Une trace persistante est ainsi générée à chaque étape, ce qui permet de reprendre l'analyse à chaque étape appliquée ou de suspendre l'analyse et de la reprendre ensuite.
- D'autres applications peuvent également être considérées : celles qui correspondent aux interactions. Dans l'état actuel de notre système, ces interactions correspondent à des traitements simples, dont la mise en œuvre repose très fortement sur la bibliothèque ILOG VIEWS. À ce titre, les interactions n'ont pas été implémentées sous forme de programmes indépendants, comme les applications de traitement d'images. Elles font partie intégrante de l'interface interactive, tout en restant isolées des autres composantes de cette interface. Nous désignons ces applications d'*applications internes*, par opposition aux applications de traitement d'images, qui sont qualifiées d'*applications externes*. Les interactions mises en œuvre sont détaillées [§ 15.6].

La progression de l'analyse peut ainsi être formalisée selon le schéma présenté [FIG. 15.1]. Ce schéma peut servir de cadre directeur pour la définition du contrôleur de dialogue, qui reste cependant réduit à sa plus simple expression. Nous avons souhaité délibérément restreindre ce contrôleur, afin de laisser la plus grande latitude possible à l'utilisateur, tout en lui présentant des choix qui restent cohérents avec les données qu'il manipule.

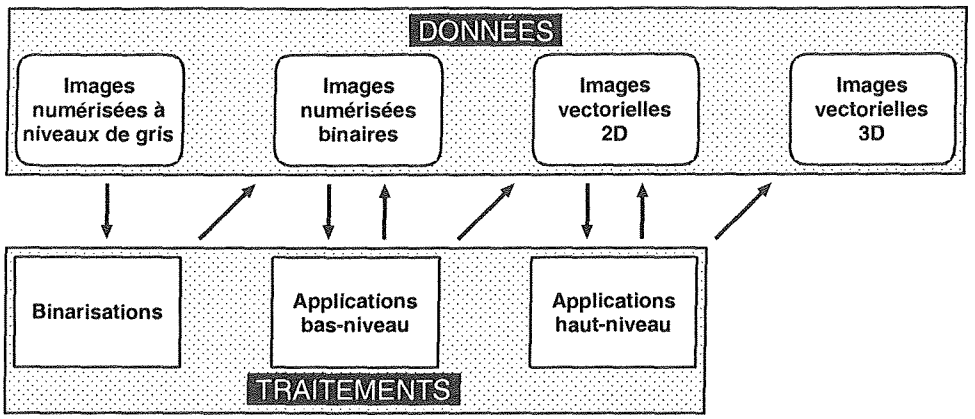


FIG. 15.1 – *Une modélisation simple de la progression de l'analyse.*

Nous avons d'autre part été attentif à la séparation de l'IHM et du cœur de l'analyse. Ces deux niveaux sont indépendants, et correspondent aux applications et à la présentation. L'interface avec les applications est réalisée au moyen d'une composante C++ qui constitue le passage obligé pour le contrôleur afin de communiquer avec les applications. En resituant les attributions des composantes de notre système interactif, on peut faire les remarques suivantes :

- l'*interface avec les applications* [§ 15.3] est implémentée par une composante C++. Cette composante, sur requête du contrôleur de dialogue, exécute les applications de la couche applicative. Lors de l'exécution des applications, elle récupère les messages générés et les transmet au contrôleur. À la fin de l'exécution d'une application, elle fournit au contrôleur une valeur caractérisant la terminaison de l'application ;
- le *contrôleur du dialogue* [§ 15.4] gère le dialogue homme-machine. Il personnalise la présentation en fonction de l'avancement de l'analyse tel que formulé [FIG. 15.1]. Suite aux interactions réalisées par l'utilisateur, reçues à partir de la présentation, il envoie des requêtes aux différentes interactions mises en œuvre ou au module d'interface avec les applications si une application externe doit être exécutée. Dans ce dernier cas, il reste en contact avec l'interface et détermine les actions à mettre en œuvre suite à la terminaison (correcte ou non) de l'application ;
- la *présentation* [§ 15.5] est réalisée par l'intermédiaire des composantes disponibles dans la bibliothèque ILOG VIEWS. Cette bibliothèque fixe un cadre de définition de cette présentation. Un souci particulier a d'autre part été porté sur l'intégration des règles ergonomiques présentées [§ 13.2.3].

15.3 Interfaçage des applications externes

Cette composante est chargée de la communication entre la deuxième couche et la troisième couche de notre système. Nous rappelons ci-dessous les choix techniques que nous avons fait :

- toutes les applications de traitement d'images sont des programmes indépendants, qui peuvent être lancés en ligne de commande,
- ces applications peuvent être considérées comme des boîtes noires : elles acceptent en entrée une ou plusieurs images ainsi que des paramètres, et fournissent en résultat une ou plusieurs images ainsi qu'un code de retour de programme indiquant les conditions d'exécution du programme (normales ou anormales).

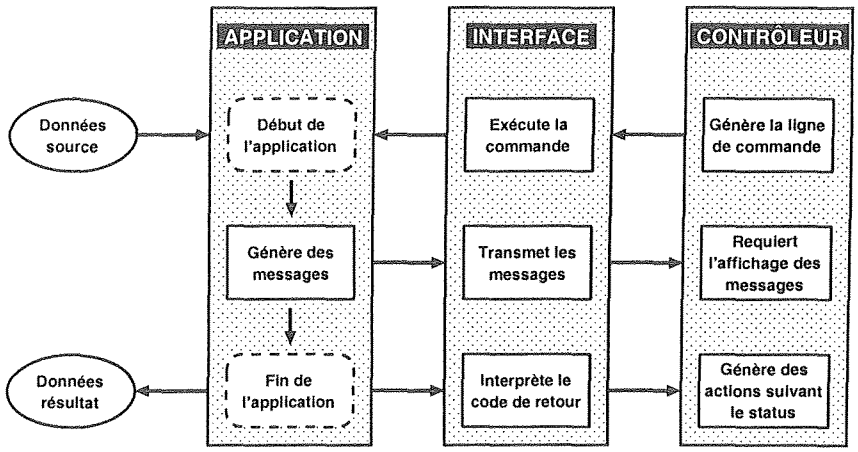


FIG. 15.2 – *Schéma des fonctions de l'interface des applications.*

Dans ce cadre, nous avons défini une interface des applications dont le principe de fonctionnement est exposé [FIG. 15.2]. L'interface permet d'exécuter des applications à partir d'une ligne de commande qui a été préparée par le contrôleur de dialogue. Les messages d'avancement générés par les applications, normalement destinés à un affichage sur la console d'où sont exécutés les applications, sont détournés par l'interface. Celle-ci les transfère alors au contrôleur, qui effectue une requête auprès de la présentation pour les afficher. Enfin, l'interface récupère le code de retour de l'application, dont le sens est interprété afin de déterminer le statut de l'application : exécution normale ou anormale. Ce statut est transmis au contrôleur. L'interface permet également de stopper l'exécution d'une application avant sa terminaison sur demande du contrôleur. Cette fonctionnalité est intéressante dans le cas où l'utilisateur ne souhaite pas mener une application à son terme.

15.4 Le contrôleur de dialogue homme-machine

Cette composante est chargée de la gestion du dialogue homme-machine. Pour cela, elle synthétise les informations qui lui sont communiquées par l'utilisateur *via* la présentation et qui lui parviennent des applications *via* l'interface. La synthèse effectuée permet ensuite au contrôleur d'effectuer des requêtes à l'interface et à la présentation [FIG. 15.3]. Les différentes phases du cycle de vie du contrôleur sont détaillées ci-dessous :

- *Collecte des informations.* Celles transmises par l'interface sont compactes et liées aux applications extérieures exécutées (messages, notification et statut de la terminaison). Les interactions peuvent engendrer un flux de communication plus important. En effet, chaque événement (clavier, souris) peut potentiellement être raccordé au contrôleur. C'est la présentation qui transmet les interactions effectuées par l'utilisateur.
- *Synthèse de l'état du dialogue homme-machine.* Le contrôleur est la composante gestionnaire de l'état de l'interaction. Cet état est caractérisé par un ensemble de variables, reflétant notamment la progression dans l'analyse (suivant l'organisation proposée [FIG. 15.1]), mais également des données qui peuvent être relatives à une interaction particulière. À partir des informations qu'il réceptionne, le contrôleur met à jour les variables d'état. Il détermine ensuite les requêtes qu'il va effectuer en comparant l'état du système à la modélisation de l'interaction. Cette modélisation est assez réduite : sa structure générale est calquée sur le schéma [FIG. 15.1] et elle se développe

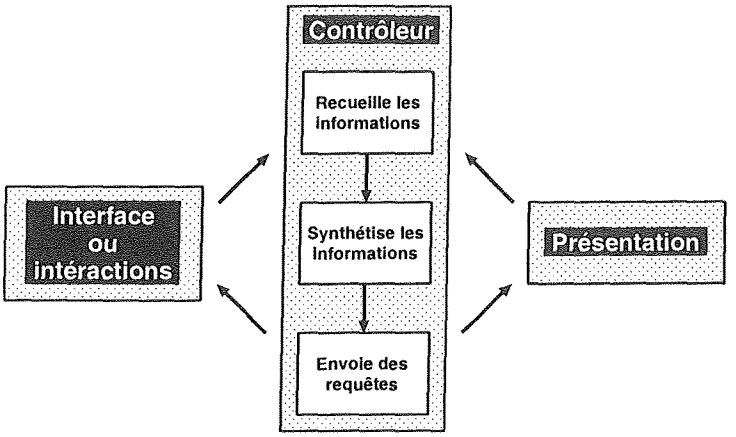


FIG. 15.3 – *Cycle de fonctionnement du contrôleur de dialogue.*

localement pour structurer les interactions mises en œuvre. Notons toutefois qu'à ce niveau de granularité, la modélisation des interactions est significativement influencée par les conditions d'utilisation des composantes ILOG VIEWS.

Dans le contexte particulier de la sollicitation d'une application extérieure *via* la présentation, le contrôleur prépare la ligne de commande destinée à l'exécution de l'application, fait transiter les messages générés par l'application de l'interface à la présentation, et effectue des actions suite à la terminaison des applications — le plus souvent, cela correspond à l'ouverture des images résultats sous l'éditeur. Le contrôleur peut également interrompre l'exécution d'un programme par requête à l'interface, suite à une demande de l'utilisateur *via* la présentation — une boîte de dialogue proposant cette fonctionnalité est visible durant toute l'exécution de l'application.

Enfin, la synthèse de l'état du dialogue inclut également une composante de vérification des erreurs engendrées par des événements extérieurs (les applications, l'utilisateur). Ceci comprend le traitement des terminaisons anormales d'application, les erreurs relatives aux fichiers, les actions inconsistantes de l'utilisateur (impression sans ouverture préalable de fichier par exemple).

- *Envoi de requêtes.* Suivant la situation, ces requêtes peuvent être destinées à l'interface ou à la présentation. Dans ce dernier cas, les requêtes permettent de synchroniser l'état effectif du système avec l'état perçu par l'intermédiaire de la présentation. Là encore, le mode de communication avec la présentation est fortement influencé par les services disponibles avec les composantes ILOG VIEWS.

15.5 Caractéristiques de la présentation

Voir [FIG. 15.4] pour une présentation de l'interface interactive développée. La partie centrale de l'interface présente l'image courante. Nous détaillons ci-dessous les différentes composantes de cette interface :

1. une barre de menus déroulants. Trois de ces menus (`Fichier`, `Édition` et `Options`) sont des menus statiques qui proposent des opérations courantes pour tout logiciel de visualisation. Le menu `Traitements` est un menu contextuel, qui présente les différents traitements applicables sur l'image courante de l'éditeur. Ce menu est personnalisé en fonction du type de l'image courante.

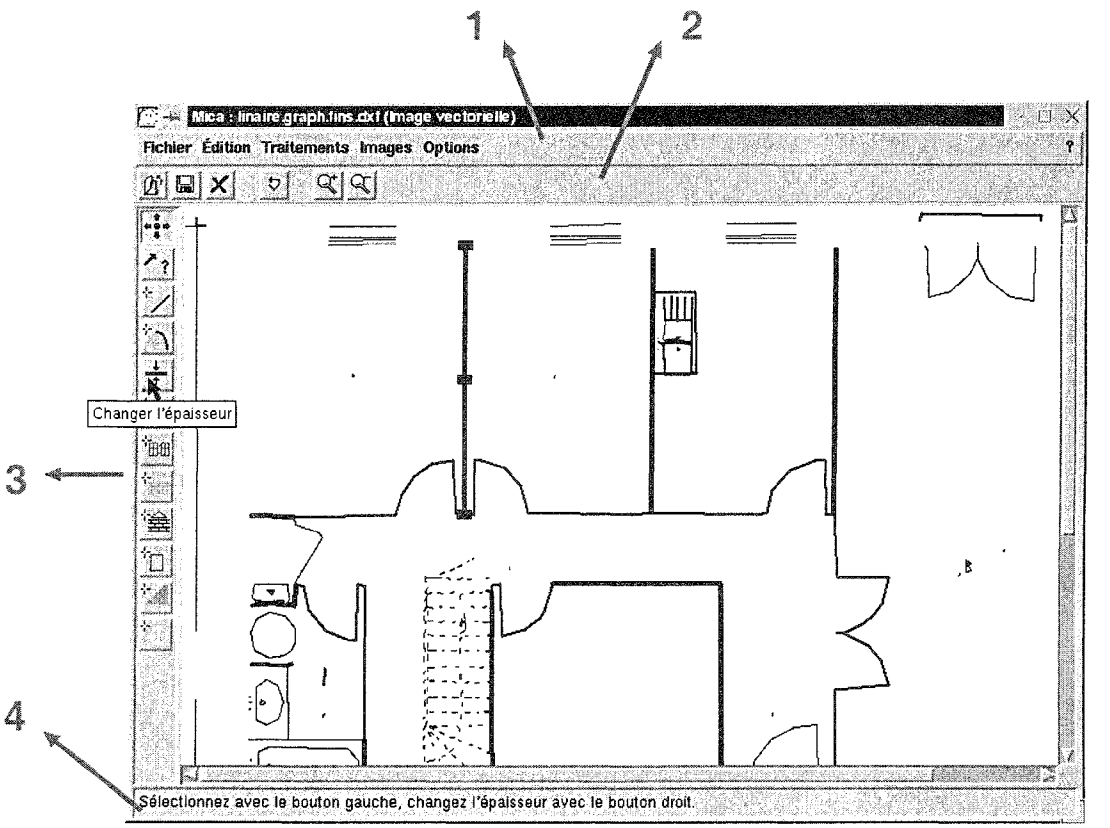


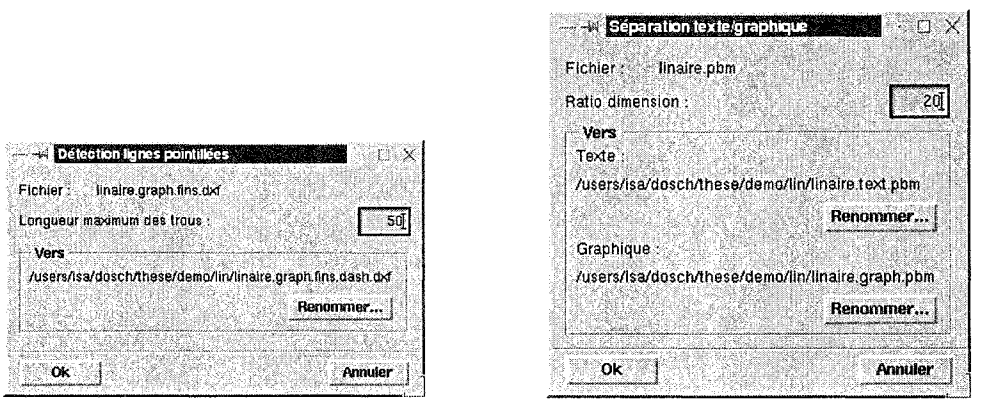
FIG. 15.4 – *La fenêtre principale de MICA.*

Le menu `Images` est un menu dynamique, permettant d'accéder rapidement à n'importe quelle image chargée sous l'éditeur ;

2. une barre d'outils statique, présente en permanence. Cette barre permet d'accéder rapidement aux opérations disponibles dans les menus ;
3. une barre d'outils contextuelle. Cette barre présente les outils d'interactions utilisables sur l'image courante, en fonction de son type ;
4. une barre d'état. Cette barre permet de présenter des informations à l'utilisateur sur les actions qu'il entreprend et d'une manière plus générale sur l'état du dialogue homme-machine. Elle met en œuvre le principe de retour d'information énoncé [§ 13.2.3].

MICA est doté de certaines fonctionnalités accessibles en permanence :

- visualisation des images chargées par l'utilisateur. À un moment donné, une seule de ces images est visible sous MICA. Elle est appelée *image courante*. Cette visualisation peut être affectée par un changement d'échelle, ce qui permet d'agrandir des détails, ou d'avoir une vue d'ensemble d'une image. Les différentes images chargées partagent le même repère géométrique et le même facteur d'échelle. Cette dernière fonctionnalité permet à l'utilisateur de se focaliser sur une portion identique des images chargées ;
- présentation de boîtes de dialogue permettant de saisir les valeurs des paramètres des différentes applications accessibles [FIG. 15.5].



(a) Détection de ligne pointillées.

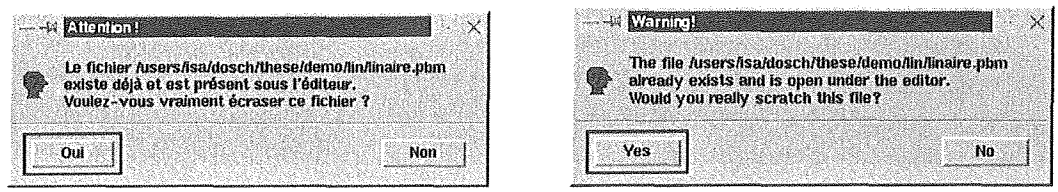
(b) Séparation texte / graphique.

FIG. 15.5 – *Exemple de boîtes de dialogue associées à des applications.*

Nous avons par ailleurs inclus certaines des considérations ergonomiques présentées [§ 13.2.3] afin d'intégrer des critères qualitatifs au niveau de la présentation. C'est en particulier le cas pour :

- *la cohérence de la présentation.* Nous avons mis l'accent sur l'uniformisation de la présentation des différentes composantes de MICA. Ce travail est facilité par l'utilisation de la bibliothèque ILOG VIEWS qui propose des primitives dont la présentation est cohérente. Nous avons maintenu cette cohérence en organisant les différentes boîtes de dialogue selon un même principe. Pour les boîtes de dialogue associées aux applications externes par exemple, l'organisation est la suivante : nom de l'image originale, les différents paramètres du traitement, nom des images résultats, et des boutons de validation ou d'annulation à une place fixe [FIG. 15.5].

- *les bulles d'aides* [FIG. 15.4]. Elles apparaissent lorsque le pointeur de souris reste immobile quelques instants sur un des outils disponibles, en affichant une description courte de l'outil dans une bulle d'aide et un mode d'emploi succinct de cet outil dans la barre d'état.
- *la représentation externe des données.* Cette fonctionnalité met en œuvre le principe de flexibilité. Tous les messages textuels utilisés sous MICA sont définis dans une base de données externe à l'interface graphique et sont disponibles en deux langues (français et anglais) [FIG. 15.6]. Il est possible, par le biais d'une commande du menu `Option` de passer de l'une à l'autre. Il est également possible, quelle que soit la langue considérée, de mettre à jour les messages sans avoir à modifier le programme de l'interface.



(a) En français. (b) En anglais.

FIG. 15.6 – *Exemple de boîtes de dialogue d'avertissement.*

15.6 Les interactions mises en œuvre

Nous détaillons ici les interactions disponibles en fonction du type d'image courante (affichée sous l'éditeur à un moment donné). Peu d'interactions sont disponibles pour les images numérisées. En effet, il existe de nombreuses applications pour traiter ce type d'images, très complètes et performantes (voir notamment GIMP [Mat 00]). Nous n'avons donc mis en œuvre que des interactions minimales pour ce type d'images.

15.6.1 Images numérisées en niveaux de gris

- **Fonction :** localisation géographique d'un point de l'image. Cette fonctionnalité est notamment utilisée pour déterminer l'épaisseur maximum des primitives graphiques.
  Présentation : affichage des coordonnées du point activé à la souris dans la barre d'état.

15.6.2 Images numérisées binaires

L'interaction présentée pour les images numérisées en niveaux de gris est également disponible pour les images binaires. Les interacteurs suivants sont également définis :

- **Fonction :** déplacement d'une zone de points d'une image vers une autre. Cette fonctionnalité permet de corriger des erreurs de segmentation (texte / graphique ou traits forts / traits fins par exemple). Les points noirs de la zone sont supprimées de l'image initiale et déplacés vers la seconde image (masque « ou logique »).

Précondition : les deux images considérées ont la même taille pour éviter les problèmes de recalage de la zone déplacée. Cette précondition est *a priori* peu contraignante, les images résultant d'une segmentation ont la même taille que l'image originale.
Présentation : après sélection d'une zone rectangulaire à la souris, une boîte de dialogue présente la liste des images vérifiant la précondition, et l'interaction est effectuée si une de ces images est sélectionnée [FIG. 15.7].

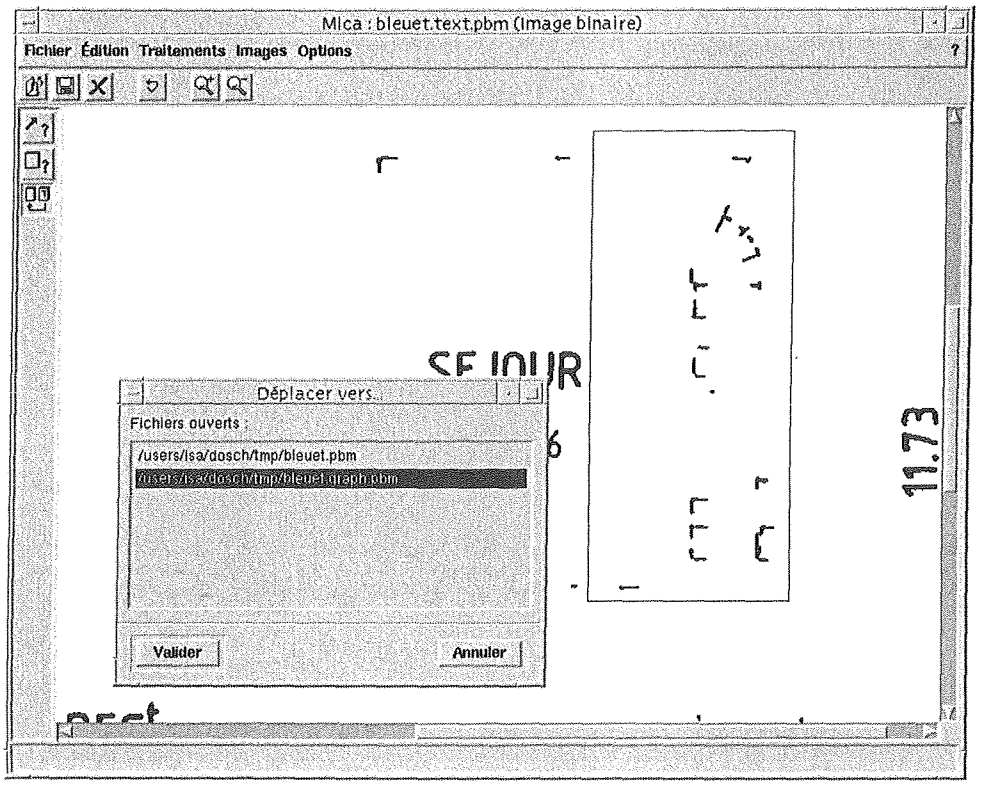


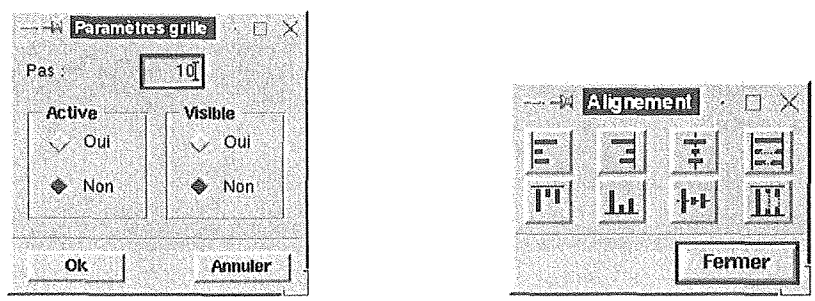
FIG. 15.7 – *Correction de la segmentation texte / graphique. Sur la couche texte, des pointillés sécants assimilés à des caractères par la segmentation automatique sont déplacés interactivement sur la couche graphique.*

- **Fonction :** édition des boîtes englobantes des chaînes de texte localisées sur une image correspondant à une couche de texte.
  Précondition : cet interacteur n'est disponible qu'à l'issue d'une extraction de chaînes telle que présentée [§ 3.4].
  Présentation : il est possible de sélectionner une ou plusieurs boîtes englobantes. Après cette sélection, l'utilisateur a la possibilité de supprimer les boîtes sélectionnées, ou de les fusionner. Il est également possible de définir directement une boîte englobante.

15.6.3 Images vectorielles

- **Fonction :** localisation géographique d'un point de l'image. Cette fonctionnalité est notamment utilisée pour déterminer l'épaisseur maximum des primitives graphiques.
  Présentation : affichage des coordonnées du point activé à la souris dans la barre d'état.

- **Fonction :** déplacement ou redimensionnement des objets graphiques. Le déplacement peut être *libre* : les objets sont alors déplacés sans aucune astreinte. Le déplacement peut également être *lié* : le déplacement d'un objet entraîne le redimensionnement des objets qui lui sont connexes pour que les extrémités de ces objets restent confondues.
  Présentation : après sélection des primitives, il est possible de les déplacer ou de les redimensionner à la souris. Le type de déplacement (libre ou lié) est sélectionné à partir de deux options exclusives du menu `Édition`.
- **Fonction :** création de primitives graphiques (arcs ou segments) ou de symboles architecturaux (7 possibilités).
  Présentation : après sélection du bouton correspondant au type de création souhaitée, la barre d'état indique la marche à suivre pour créer l'objet à la souris.
- **Fonction :** modification de l'épaisseur d'une primitive graphique.
  Présentation : après sélection du bouton correspondant au type de création souhaitée, la barre d'état indique la marche à suivre pour modifier l'objet [FIG. 15.4].
- **Fonction :** utilisation d'une grille pour contraindre la position des objets graphiques selon un certain pas, exprimé en *pixels*.
  Présentation : une boîte de dialogue accessible dans le menu `Édition` permet de sélectionner ou de désélectionner cette fonction, de préciser la valeur du pas et la visibilité de la grille [FIG. 15.8(a)].



(a) Paramètres de la grille.

(b) Alignement d'objets.

FIG. 15.8 – *Boîtes de dialogues associées aux interactions.*

- **Fonction :** alignement des objets graphiques entre eux. Ces alignements permettent, selon chacun des axes, de centrer les objets ou de les aligner selon une des quatre directions : droite, gauche, haut ou bas.
  Présentation : après sélection d'un premier objet, faisant office de référence, et des objets à aligner par rapport à l'objet référence, une boîte de dialogue (accessible dans le menu `Édition`) permet de sélectionner le type d'alignement souhaité [FIG. 15.8(b)].

15.7 Conclusion et perspectives

L'objectif principal de la mise en œuvre de cette interface est de faciliter le dialogue homme-machine. En adoptant des principes de coopération homme-machine et d'architecture logicielle, nous avons défini

un environnement complet d'analyse de documents techniques dont l'interface interactive n'est que la partie visible. Dans l'état actuel de son développement, on peut formuler les remarques suivantes sur MICA :

- MICA apporte une facilité d'utilisation de toutes les applications de traitement d'images que nous avons mis en œuvre. Il est en effet plus aisé pour l'utilisateur de superviser les différentes étapes nécessaires. MICA offre de plus certaines interactions permettant de modifier les images manipulées, ce qui n'aurait pas été possible en dehors de tout environnement graphique.
- L'architecture logicielle adoptée, dont la qualité principale est la modularité, fait de MICA, et de notre système d'analyse plus généralement, un système facilement maintenable et extensible. Cet aspect est primordial dans le développement de logiciels et encore plus particulièrement dans notre contexte. En effet, il apparaît clairement que la vocation de ce système est de représenter de façon permanente l'état d'avancement de nos travaux en analyse de documents techniques. Cette contrainte n'est réalisable que si elle est prise en compte dès la conception du système par des procédés *ad hoc*, tels que ceux exposés dans cette dernière partie.

Tout en constituant un environnement pratique, réutilisable dans d'autres domaines de recherche, il serait intéressant de compléter les fonctionnalités offertes par MICA afin d'améliorer la qualité de son utilisation ou de sa maintenance :

- Par des modèles de documents : les boîtes de dialogue associées aux différentes applications externes présentent toujours des valeurs par défaut pour les paramètres de ces applications. On constate que les documents manipulés constituent parfois des « familles », c'est-à-dire des ensembles de documents présentant des propriétés communes (bruit, résolution, style architectural). La notion de modèle de document pourrait permettre de rassembler la configuration de ces différents paramètres, ce qui permettrait de les mettre à jour simultanément par le biais d'une seule opération.
- Par des macro-commandes : cette fonctionnalité est intéressante dans le contexte des familles de documents, mais également dans le contexte du tuilage. L'objectif serait par exemple d'enregistrer par un langage de macro-commandes les applications, ainsi que les paramètres qu'elles nécessitent, lors du traitement de la première tuile pour appliquer ensuite ces mêmes traitements aux autres tuiles. Bien entendu, cette fonctionnalité n'est que pleinement appréciable sur des documents qui nécessitent peu d'interactions, c'est-à-dire des documents peu bruités et dont la modélisation est proche d'un enchaînement des traitements que nous avons développé.
- Par d'autres représentations externes de données. La représentation externe est déjà utilisée dans notre interface interactive pour la manipulation des composantes textuelles. Il serait très intéressant d'étendre ce type de représentation à d'autres composantes, comme les boîtes de dialogue associées aux applications externes. Nous avons souligné dans ce chapitre l'effort d'uniformisation de présentation que nous avons entrepris pour ces composantes. Cet effort correspond à un premier pas vers une formalisation de la description de ces boîtes. Cependant, afin d'être complètement intéressante, la formalisation devrait être étendue aux autres caractéristiques liées aux applications externes : leur nom, les paramètres attendus, le type d'image auquel elles s'appliquent... L'idéal serait dans ce contexte de créer une ressource associée à chacun de ces traitements, contenant ces différentes informations. Cela permettrait de ne pas avoir à modifier du tout le programme de l'interface pour ajouter un nouveau traitement et améliorerait encore sensiblement sa maintenance et son extensibilité.
- Par l'utilisation d'autres boîtes à outils. Si les boîtes à outils permettent de disposer de nombreux avantages relatifs à la structuration des composantes, à l'ergonomie, à l'intégration de principes de coopération homme-machine, elles contraignent également fortement les possibilités de dialogue

homme-machine aux principes qu'elles intègrent. Après avoir utilisé ILOG VIEWS pour le développement de MICA, nous réfléchissons actuellement à l'utilisation de nouvelles boîtes à outils, plus proches de nos attentes. Cela pourrait être en particulier le cas avec des outils tels que QT ou KDE [Dal 99] qui proposent des protocoles de communication entre les composantes graphiques *a priori* plus intéressants que ceux disponibles sous ILOG VIEWS. L'annexe A présente un état des fonctionnalités, des avantages et des inconvénients, que nous avons relevé lors de l'utilisation d'ILOG VIEWS.

Conclusion et perspectives

Dans cette thèse, nous avons montré la faisabilité d'une approche automatisée — l'homme restant dans la boucle, mais intervenant le moins possible — de la modélisation 3D de bâtiments à partir des plans numérisés de type avant-projet. Pour cela, nous avons développé un environnement complet d'analyse de plans architecturaux, composé de trois couches. La première couche est une bibliothèque de traitements de bas et moyen niveau pour la reconnaissance de graphiques, comprenant des méthodes qui représentent l'état de l'art actuel dans ce domaine. La deuxième couche correspond à un ensemble d'applications indépendantes qui permettent de mener l'analyse de plans architecturaux. À ce niveau, aucun enchaînement n'est imposé, permettant une bonne flexibilité face aux diverses représentations possibles des plans architecturaux. Enfin, la troisième couche est une interface interactive, qui permet à l'utilisateur de superviser l'analyse et d'interagir avec les données. Cette couche permet d'exécuter les applications de la deuxième couche. L'architecture logicielle proposée avec le couplage de la bibliothèque ISADORA et l'interface MICA donne une grande souplesse et une bonne adaptabilité à l'évolution des données techniques, tout en facilitant l'interaction homme–machine à tous les stades du processus d'interprétation.

Au niveau des méthodes mises en œuvre, nous avons différencié les méthodes bas-niveau, s'appliquant sur les images numérisées, des méthodes haut-niveau, manipulant des données vectorielles.

- La majeure partie des méthodes bas-niveau sont des méthodes matures, éprouvées, qui correspondent à l'état de l'art de notre domaine. Les résultats obtenus avec ces méthodes sont généralement de bonne qualité et nécessitent peu d'intervention humaine, que ce soit pour le réglage de leurs paramètres ou pour les interactions nécessaires à la correction des résultats. On peut noter cependant que, même à ce niveau, des problèmes restent ouverts, comme le problème de la segmentation texte / graphique, « classique » en analyse de documents.
- Nous avons vu que l'étape de *vectorisation*, permettant de convertir les images numérisées en images vectorielles, demeure une préoccupation pour notre équipe et pour les chercheurs du domaine en général. Beaucoup de méthodes ont été proposées pour traiter ce problème délicat. Si certaines méthodes *ad hoc* permettent d'obtenir de bons résultats, peu de solutions sont satisfaisantes d'un point de vue générique. Il reste donc *a priori* des pistes à explorer pour améliorer cette étape, notamment au niveau de la précision des résultats obtenus. Les traitements accompagnant le processus même de vectorisation, comme la détection d'arcs de cercle ou de lignes pointillées, sont relativement robustes, à partir du moment où la densité des données est suffisante. Jusqu'à ce point, les méthodes mises en œuvre ne sont pas spécifiques aux plans d'architecte et peuvent être appliquées à d'autres types de documents techniques.
- Enfin, les méthodes de haut-niveau sont plus spécifiquement destinées à l'analyse de plans architecturaux, même si elles peuvent être facilement adaptées à d'autres types de documents techniques. L'analyse de symboles architecturaux correspondant à des menuiseries peut en particulier, de par la description externe des symboles à rechercher, être utilisée à d'autres fins. Il en est de même pour la recherche des cages d'escalier, dont l'étape de détection de texture peut être

réutilisée pour d'autres types de textures. Bien que relativement rudimentaire, la technique de reconstruction par extrusion de chaque étage donne des résultats satisfaisants, et permet une mise en correspondance robuste des différents étages, dans les types de cas testés. Enfin, les processus de mises en correspondance (pour le tuilage et le recalage de niveaux) mis en œuvre sont *a priori* assez robustes dans le cadre de notre utilisation.

Il reste néanmoins un certain nombre de limitations aux résultats actuels ; certaines sont inhérentes à l'état de l'art dans le domaine, d'autres nécessiteraient des développements et des expérimentations supplémentaires.

- Dans les outils de bas niveau, nous restons relativement peu satisfaits de la stabilité et de la robustesse des outils de binarisation adaptative dans le cas de plans présentant des pliures, des taches, et autres bruits sensibles dans l'image à niveaux de gris. Il reste ici un travail de fond à faire sur la mise au point de filtres de binarisation adaptés au cas des plans techniques en mauvais état, l'essentiel de l'état de l'art dans le domaine portant sur le domaine connexe, mais néanmoins différent, de la reconnaissance des écritures sur des chèques.
- Comme nous l'avons vu, la vectorisation est un élément central du processus d'analyse et de reconstruction. Bien que nous ayons une méthode qui correspond à l'état de l'art dans le domaine, il reste des travaux à réaliser, notamment sur un meilleur positionnement des jonctions et une meilleure intégration entre arcs et segments. Plusieurs pistes sont actuellement à l'étude dans la communauté scientifique [Jan 97b, Röö 96], et notre équipe compte s'investir dans les années à venir sur ce problème difficile.
- Si nous reconnaissons actuellement correctement un certain nombre d'indices présents dans les plans architecturaux, et notamment les lignes tiretées et les symboles de menuiseries, d'autres indices nécessitent encore des développements, notamment les hachurages (en particulier pour détecter les murs porteurs représentés non pas par des traits forts, mais par des rectangles hachurés), et les cages d'escalier, pour lesquelles la méthode actuelle reste assez heuristique.
- La modélisation 3D par extrusion donne des résultats satisfaisants pour le but recherché dans cette étude, sur les exemples testés, mais la combinaison de toutes les limitations indiquées ci-dessus rend pour l'instant difficile le passage à l'échelle, c'est-à-dire le traitement de plans de plus grande taille et de plus grande complexité. Il y a là indéniablement matière à un développement supplémentaire, mais celui-ci est fortement dépendant de l'amélioration de la qualité et de la robustesse des outils de bas et moyen niveau.
- Un autre passage à l'échelle délicat est celui d'une exigence de précision plus grande dans les modélisations 2D et 3D. Ce problème peut être envisagé par la consolidation des outils existants, et par l'adjonction de techniques supplémentaires telles que l'analyse de la cotation [Col 92] et la mise en correspondance entre les plans actuellement traités et des vues de face ou des coupes, par exemple.

Suite à ces travaux, et à certaines des limitations évoquées, nous pensons qu'il est intéressant de développer ce travail à plusieurs niveaux.

- Il est nécessaire de procéder à une évaluation de performances plus accrue. Notre communauté scientifique met actuellement l'accent sur la nécessité de procéder à des évaluations de performances, afin de mieux maîtriser l'état de l'art de notre domaine. L'approche la plus facile pour y arriver consiste à élaborer des bases de données communes et à utiliser une sorte de mesure pour les comparer [Chh 99].
- Il reste des améliorations à apporter aux traitements existants. Dans un premier temps, nous souhaitons nous concentrer sur les problèmes de bas-niveau (binarisation, vectorisation, traitements

des hachures), qui influent directement sur les étapes haut-niveau. Nous souhaitons notamment améliorer la qualité des résultats et étudier les problèmes de précision.

- Après avoir conçu l'intégralité de notre système, nous souhaitons également améliorer l'architecture logicielle. Il serait par exemple intéressant de pouvoir spécifier les interactions de façon extérieure, à la manière des applications de traitement d'images, afin de renforcer encore la modularité du système. Dans une certaine mesure, nous souhaitons également nous tourner vers une représentation externe accrue, en particulier sur les boîtes de dialogues correspondant aux applications externes. L'objectif serait de permettre une mise à jour dynamique de l'interface interactive lors de l'insertion ou la modification d'un traitement. Nous souhaitons également nous intéresser à d'autres boîtes à outils logicielles, plus adaptées aux dernières évolutions de la coopération homme-machine.
- Enfin, nous étudions la possibilité de faire une mise à disposition auprès de la communauté scientifique de notre système. Ce type de diffusion, très en vogue depuis l'essor des logiciels libres, cumule plusieurs avantages. Il permet tout d'abord de faire profiter la communauté scientifique des algorithmes et des implémentations réalisés par l'équipe, et non plus seulement des méthodes. Cette interaction avec la communauté permet d'une part d'obtenir un retour plus réactif sur la qualité des méthodes et de leur mise en œuvre, et d'autre part de fédérer les chercheurs intéressés autour d'un système commun, comme cela est le cas avec des environnements tels que MDUS.

Annexe A

Ilog Views

A.1 Présentation d'Ilog Views

Ilog Views est une boîte à outils[47] permettant de créer des applications graphiques dotées d'interfaces interactives. Ilog Views est constitué de plusieurs bibliothèques de classes C++, correspondant aux différentes composantes graphiques proposées, ainsi que d'un logiciel, *ivstudio*, permettant de construire interactivement des interfaces et dans une certaine mesure d'associer des actions aux composantes de ces interfaces[48]. Les composantes graphiques d'Ilog Views permettent de créer des interfaces graphiques allant d'une simple « fenêtre système » (X ou Windows) à des applications plus élaborées, comportant plusieurs fenêtres, des zones de saisie, des courbes résultat, des primitives graphiques (points, segments, splines...) et des interactions avec l'utilisateur.

Le choix de cette boîte à outils a été influencé par le cahier des charges défini dans le cadre du contrat avec le CNET. Ce choix a été *a priori* raisonnable : Ilog Views offre une grande gamme de classes C++ prêtes à l'emploi, permettant de gérer aussi bien les interfaces graphiques nécessaires pour la construction de boîtes de dialogue, que les primitives graphiques nécessaires à la représentation des données vectorielles. Enfin, le caractère extensible et portable d'Ilog Views constituent des atouts indispensables pour le développement d'une application telle que MICA.

A.2 Les fonctionnalités

A.2.1 Les objets graphiques

Les *objets graphiques* correspondent aux différentes entités graphiques disponibles sous Ilog Views. Toute une hiérarchie d'objets est définie, allant des plus simples (point, segment, arc, ...) aux plus compliqués (boîte de dialogue prédéfinie par exemple). Tous ces objets dérivent d'une base commune (la classe `IlvGraphic`) et partagent ainsi certaines caractéristiques :

- ils possèdent une *bounding box* qui détermine leur emplacement et leurs dimensions (largeur et hauteur) ;

47. Selon la terminologie introduite [§ 13.3.4].
48. Ce qui permet de prototyper assez rapidement des interfaces interactives.

- les objets sont dotés de méthodes permettant leur affichage, avec une *transformation* éventuelle (translation, rotation, redimensionnement). Cette transformation est alors définie grâce à une instance de la classe `IlvTransformer` ;
- leurs caractéristiques géométriques (position, dimensions) sont modifiables ;
- leurs caractéristiques graphiques (couleur par exemple) sont modifiables. Ces caractéristiques sont appelées *ressources* ;
- ces objets peuvent être lus ou écris par l'intermédiaire de flux (*streams*) spéciaux ;
- des propriétés (utilisateur) supplémentaires peuvent facilement être ajoutées ;
- enfin, certains renseignements les concernant (leur nom, leurs classes de base) sont accessibles.

Une petite partie de la hiérarchie développée à partir de la classe `IlvGraphic` est présentée figure A.1. Les classes dérivées de la classe `IlvSimpleGraphic` sont présentées dans un premier temps, et nous détaillons plus spécialement ensuite celles issues de la classe `IlvGadget`.

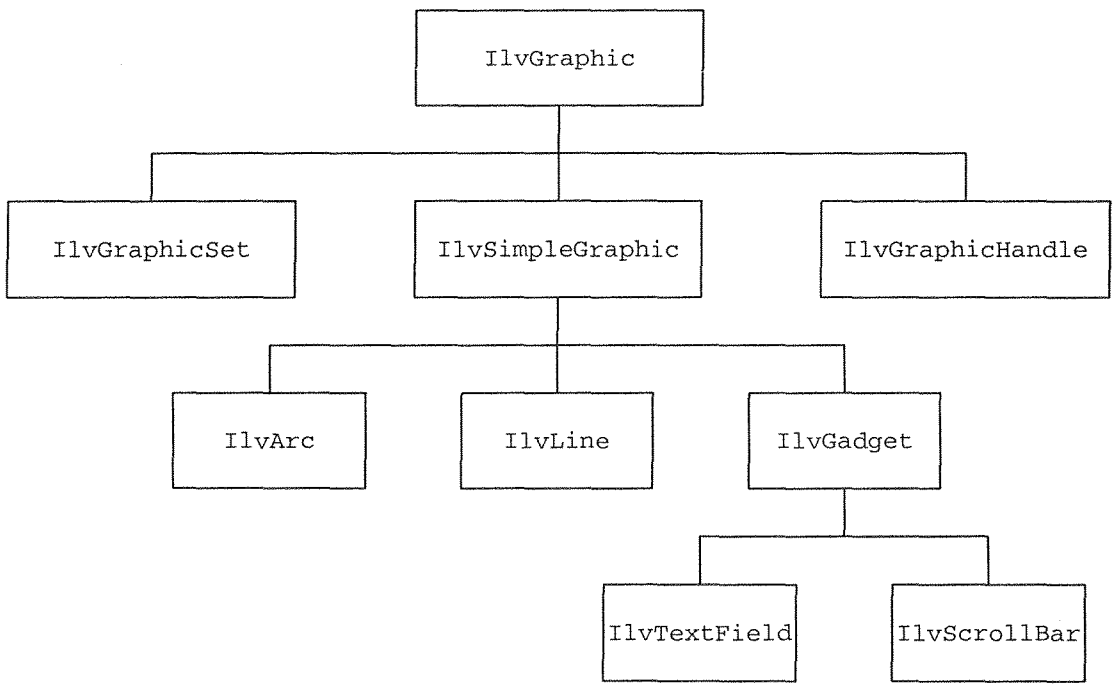


FIG. A.1 – *La classe* `IlvGraphic` *et une partie de ses classes dérivées.*

A.2.2 Les objets graphiques de base

Ces objets correspondent à des classes dérivées de la classe `IlvSimpleGraphic`. Ils permettent de représenter les primitives graphiques de base, comme les segments et les arcs, et peuvent être utilisés pour construire des objets plus évolués, comme les différents symboles architecturaux. Voici une liste non exhaustive des différents objets prédéfinis sous Ilog Views : les arcs, les ellipses, les segments, les flèches, les polygones, les images *bitmap*, les chaînes de caractères, les splines, les jauges, les gadgets...

A.2.3 Les gadgets

Certains objets graphiques prédéfinis disposent de méthodes supplémentaires permettant de réagir assez finement aux événements utilisateurs. Ces objets particuliers sont appelés *gadgets* et dérivent de la classe `IlvGadget`. Ils sont à rapprocher de la notion de *widget* telle qu'elle est définie sous Motif ou sous Windows, tout en y apportant des améliorations.

De nombreuses méthodes virtuelles de la classe `IlvGadget` sont associées à des événements tels que la prise ou la perte du *focus*, c'est-à-dire au fait que les gadgets soient sensibles ou non aux événements utilisateur à un instant donné. Ces méthodes, appelées fonctions *callback* sont facilement redéfinissables lors de dérivations des classes représentant ces gadgets.

Les gadgets prédéfinis sous Ilog Views permettent de composer, entre autres, des fenêtres de dialogues nécessaires aux interactions homme-machine. On dispose à cet effet de composantes maintenant « classique » dans toute boîte à outils : des zones de texte affichable, de zones de saisie, des boîtes à options, des boîtes à liste, des boutons, des ascenseurs, des matrices, des tableurs, des menus et des boîtes de dialogues prédéfinies. Les gadgets ont généralement un comportement par défaut. Ainsi, les boutons (classe `IlvButton`) s'enfoncent lorsque l'utilisateur les valide par l'intermédiaire du clavier ou de la souris. Ce comportement peut être modifié par dérivation de la classe décrivant le gadget.

A.2.4 Les ressources

Sous Ilog Views, les attributs des objets graphiques sont définis par des *ressources* (classes dérivées de la classe `IlvRessource`). Parmi ces ressources, on trouve : les couleurs, le style de trait (continu, pointillé, etc.), l'épaisseur du trait, la police utilisée, le *pattern*, le style de remplissage... Toutes ces ressources sont regroupées au sein d'un objet de type `IlvPalette`, qui peut être partagé entre plusieurs objets ayant dans ce cas les mêmes ressources.

A.2.5 Les vues

La *vue* sous Ilog Views (classe `IlvAbstractView` et dérivées) permet de visualiser les différents éléments d'une application Ilog Views, typiquement les objets graphiques. Dans sa forme la plus simple, elle ne permet que de visualiser et pas de stocker ces objets. Elle doit dans ce cas être utilisée conjointement avec une classe permettant le stockage et la gestion des objets graphiques. Une des plus importantes classes qui en dérive est la classe `IlvView` puisqu'elle correspond à un espace visible à l'écran. D'une manière générale, la présentation d'une application Ilog Views correspond à une ou plusieurs vues de types différents. Ces différents types de vues sont représentées sur l'arbre d'héritage de la figure A.2.

On distingue deux grandes catégories de vues : les vues top-niveau et les vues bas-niveau. Les vues top-niveau (classe `IlvView`) correspondent aux fenêtres principales visibles sur un écran (fenêtre X sous UNIX) et servent essentiellement à contenir les vues bas-niveau. Les vues haut-niveau contiennent un titre, détermine la taille maximale des fenêtres qu'elles contiennent et peuvent se faire associer un menu système. Dans la deuxième catégorie de vues, les bas-niveau, on trouve les classes principales suivantes :

- `IlvScrollView` : cette vue permet de doter l'unique vue qu'elle contient d'ascenseurs. La vue contenue est généralement plus grande que sa mère ;
- `IlvView` : vue de travail dont une partie seulement est visible à un instant donné (à utiliser conjointement avec les `IlvScrollView`) ;
- `IlvElasticView` : cette classe correspond à un type de vue dont la taille peut évoluer ;

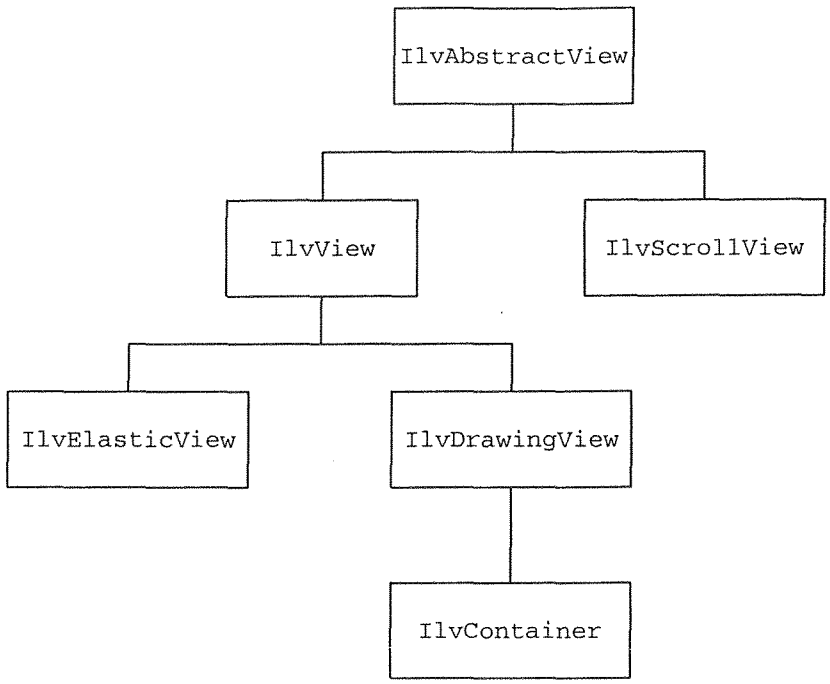


FIG. A.2 – *Arbre d'héritage des vues (simplifié).*

- `IlvDrawingView`: vue dotée de méthodes permettant d'exploiter les événements extérieurs ;
- `IlvContainer`: vue permettant de stocker des objets graphiques. Cette classe est la première dans l'arbre d'héritage des vues conjuguant le stockage et l'affichage des objets graphiques.

Remarque : la classe `IlvView` se trouve cataloguée à la fois dans les classes top-niveau et bas-niveau. Elle permet en effet d'instancier ces deux types de vues ; la nuance se fait lors de l'appel du constructeur.

A.2.6 Les containers

Les containers (instances de la classe `IlvContainer` et dérivées) sont des vues particulières qui permettent d'afficher et de stocker des objets graphiques. Ils disposent donc de nombreuses méthodes pour manipuler les objets contenus et de certains mécanismes permettant leur gestion, comme : l'affichage, le déplacement, le redimensionnement, l'agrandissement ou la réduction (*zoom*), la possibilité d'utiliser un affichage géré grâce à un double buffer...

On y trouve également développé deux grandes fonctionnalités : les *interacteurs d'objets* et les *accélérateurs*. Ces notions (développées ci-dessous) permettent de traiter les événements utilisateurs au niveau des objets graphiques, y compris ceux de base. Elles ne sont cependant pas incompatibles avec les gadgets et la gestion des événements qui leur est propre.

Les interacteurs d'objet graphique

Les interacteurs d'objet graphique permettent de filtrer les événements utilisateurs (clavier, souris) au niveau de l'objet auquel ils sont rattachés. Un interacteur d'objet peut ainsi être activé lors de la présence de la souris sur l'objet considéré, ou à travers une séquence clavier, après sélection préalable.

On peut, par ce moyen, associer une action particulière au niveau d'un objet à une série d'événements. On trouve sous Ilog Views un grand nombre d'interacteurs d'objets prédéfinis permettant, entre autres, le déplacement, le redimensionnement, ...

Les accélérateurs

Les accélérateurs permettent de filtrer les événements utilisateurs d'une façon générale (et non pas au niveau d'un objet) et d'y associer un appel de fonction. On peut donc associer une action à une séquence de touches. On trouve dans les containers un certain nombre d'accélérateurs définis par défaut, qu'il est possible de reconfigurer.

Les vues rectangles

Les vues rectangles ne sont en fait pas des vues mais des gadgets contenant une vue ; ce sont donc des objets graphiques qui s'affichent d'eux-mêmes dans une fenêtre sans avoir à passer par une vue ou un container supplémentaire. Ilog Views contient quelques vues rectangles prédéfinies bien utiles, incluant des zones de dialogue basiques ou prédéfinies, affichant un message, une question, un avertissement, une erreur ou permettant de saisir facilement le nom d'un fichier, une chaîne de caractères ou de choisir une police de caractères.

A.2.7 Les managers

Les containers permettent de stocker et gérer des objets graphiques avec une certaine facilité, mais restent cependant limités quant à leurs possibilités ; ils sont ainsi mal adaptés à la gestion d'un trop grand nombre d'objets graphiques. Afin de coordonner facilement une grande quantité d'objets graphiques, Ilog Views dispose de *managers*, qui correspondent à la classe `IlvManager` et ses dérivées. Ceux-ci sont destinés au stockage et à l'organisation des objets graphiques sans se soucier de leur affichage. Il faut donc utiliser conjointement les managers avec au moins une vue pour visualiser les objets contenus. Ces managers offrent de nouvelles fonctionnalités.

Les couches

Les managers permettent d'organiser les objets graphiques qu'ils contiennent en différentes couches. Ces couches définissent des niveaux de priorité : les objets contenus dans les couches supérieures sont affichés devant les objets contenus dans les couches inférieures.

Les interacteurs

Les managers permettent également de définir des interacteurs. Nous ne sommes cependant plus limités aux interacteurs d'objet ; les managers permettent en effet de définir des interacteurs globaux rattachés à une vue sur le manager. Ces interacteurs sont appelés *interacteurs de vue*. Lors de la réception d'événements utilisateur, les interacteurs de vue sont prioritaires sur les interacteurs d'objets.

Les accélérateurs

La notion d'accélérateur reste, elle, inchangée par rapport aux containers. La liste des accélérateurs prédéfinis s'enrichit cependant de nouveaux éléments : opérations de sélection d'objets, de gestion d'historique des commandes, ...

Autres caractéristiques

Les managers sont dotés des mêmes facilités que les containers en ce qui concerne la gestion des objets, leurs déplacements, leurs transformations géométriques, etc. Mais ils permettent également de :

- partager des objets graphiques entre plusieurs vues ;
- rattacher facilement des propriétés supplémentaires aux objets graphiques manipulés ;
- d'appliquer facilement une série d'opérations sur un objet, un groupe d'objets, les objets sélectionnés ;
- gérer un historique des commandes effectuées sous le manager, avec des possibilités d'annulation (*undo*) et de répétition (*redo*) ;
- gérer une grille (*grid*) qui aligne les événements souris sur des positions prédéfinies.

A.2.8 En résumé...

On retrouve figure A.3, les classes détaillées, classées thématiquement :

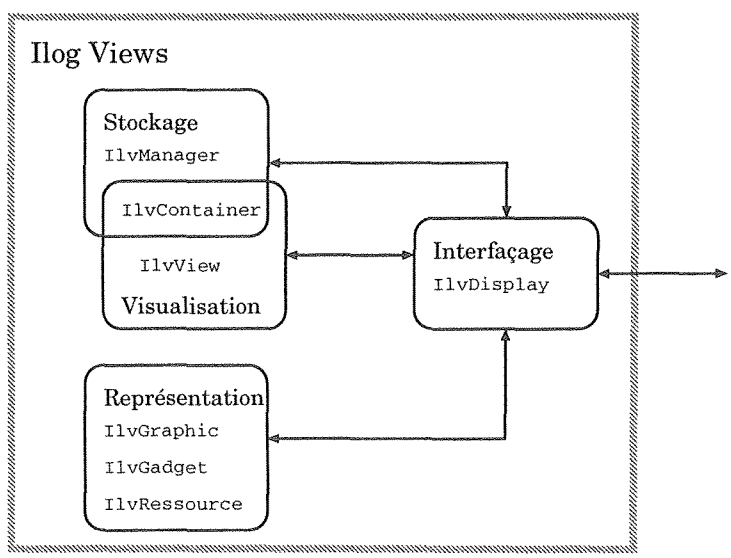


FIG. A.3 – *Synopsis des classes de base Ilog Views.*

- interfaçage avec le gestionnaire graphique du système (Windows ou X) : `IlvDisplay`,
- représentation : `IlvGraphic`, `IlvGadget`, `IlvRessource`,
- visualisation : `IlvView`, `IlvContainer`,
- stockage : `IlvContainer`, `IlvManager`.

Remarque : typiquement, en matière de stockage d'objets graphiques, les containers sont plutôt utilisés pour stocker les gadgets et les managers sont plutôt utilisés pour stocker les autres types d'objets graphiques.

A.3 Architecture générale de MICA

Le principe général d'utilisation d'Ilog Views est la dérivation de classe. En dérivant les classes disponibles, on peut en effet bénéficier des services offerts par ces classes tout en les personnalisant. Ce principe, bien connu en conception objet, est à la base de l'utilisation des différentes composantes Ilog Views. En particulier, lors de la conception d'interfaces graphiques à l'aide de l'outil interactif *ivstudio*, une classe définissant les différents gadgets définis, et dérivant d'une des classes Ilog Views, est générée. Cette classe peut alors à nouveau être dérivée pour implémenter les actions à associer aux interactions réalisées sur ces gadgets. Cette organisation présente un avantage : les gadgets de l'interface peuvent ensuite être déplacés ou personnalisés sous *ivstudio* sans avoir à modifier le code source qui implémente les actions associées à ces gadgets. Mais elle présente aussi un inconvénient : elle oblige quasi-systématiquement à définir 2 classes pour chaque interface définie, même si cette interface ne contient qu'un seul gadget.

En suivant ces principes, l'architecture générale de MICA s'articule autour des composantes suivantes :

- une classe correspondant à la fenêtre principale de l'application, dérivée de la classe `IlvContainer`. Cette classe contient une barre de menus, deux barres de boutons, la barre « statique » et la barre personnalisable, une zone où s'affichent les messages et une *vue rectangle* qui permet de stocker les différentes données manipulées (image numérisées et objets vectoriels) ;
- une classe par boîte de dialogue nécessaire (généralement associée à une application externe), ces boîtes de dialogue étant accessibles par les menus de la fenêtre principale.

La communication entre les différentes composantes Ilog Views s'effectue *via* des fonctions *callback*, ce qui oblige à définir une nouvelle fonction pour implémenter chaque nouvelle communication. Enfin, les différentes données manipulées sont également gérées à partir des composantes Ilog Views. Si les images numérisées sont directement manipulables à partir de ces composantes, il a été nécessaire de dériver les classes Ilog Views pour implanter les caractéristiques correspondant aux données vectorisées.

A.4 Les limites observées dans le cadre de notre utilisation

À l'utilisation, Ilog Views s'avère un outil puissant, permettant d'implémenter de façon structurée les différentes composantes de l'interface définie, surtout au niveau de la définition des interfaces interactives. En particulier, l'outil *ivstudio* a été très utile pour concevoir interactivement les différentes interfaces disponibles sous MICA, ainsi que pour les maintenir, pour les raisons évoquées ci-dessus. Dans le cadre de notre utilisation, nous avons cependant également relevé quelques limites :

- La première, déjà été évoquée ci-dessus, réside dans la multiplication de classes nécessaires à la définition des interfaces. La maintenance de ces classes s'avère parfois assez coûteuse par rapport aux services offerts.
- La seconde est liée aux aspects de communication entre composantes, communications basées sur les fonctions *callback*. Le principe de ces méthodes, hérité des bibliothèques telles que Motif, est

basé sur la manipulation de pointeurs de fonction. Afin de bénéficier de ce mécanisme quelles que soient les classes en présence, le type de fonctions pointées se confine aux fonctions statiques, ce qui interdit toute référence à un objet donné. Ce mécanisme de communication n'est donc pas totalement compatible avec les principes de programmation par objet et induit la nécessité de définir ces fonctions *callback*, correspondant à des fonctions C « classiques », qui « alourdissent » le code source. Sur cet aspect, de nouveaux modes de communication entre composantes, comme les *signaux* et les *slots* introduits dans la bibliothèque QT [Dal 99], sont nettement plus flexibles et conformes aux principes de programmation objet.

- Enfin, une limitation est imputable aux primitives graphiques (segments et arcs) disponibles sous Ilog Views. Si ces primitives sont disponibles de façon native, elles ne sont cependant pas gérées de façon optimale. En particulier, l'épaisseur que l'on peut associer aux primitives n'est quasiment pas prise en compte dans le processus d'affichage, ce qui oblige à redéfinir ces primitives en les dérivant. Cette opération occasionne des chutes de performances au niveau de la visualisation qui sont sensibles sur les images comprenant un grand nombre de vecteurs. Sur cet aspect, un outil spécifiquement dédié à la gestion de primitives vectorielles, comme AutoCAD, aurait sans doute été plus approprié.

Annexe B

Structure de baquets

Les structures de baquets, largement utilisées dans les algorithmes à base de calcul géométrique, permettent d'améliorer sensiblement les performances de ce type d'algorithmes [Asa 85]. Elles sont utilisées pour la gestion et la recherche d'objets situés dans une région originale ; les *baquets* correspondent alors aux sous-régions résultant du partitionnement de la région initiale. De nombreux travaux décrivent l'utilisation de ce type de structures dans le contexte des algorithmes géométriques, comme par exemple les diagrammes de Voronoï. Si le principe de ce type d'approche est bien établi, la décomposition d'une région initiale en sous-régions et la définition de méthodes pour effectuer la recherche de données stockées dans ces sous-régions, a engendré de nombreuses déclinaisons adaptées à des problèmes spécifiques [Asa 85]. Il est ainsi possible de définir :

- des baquets à forme simple, c'est-à-dire carrés ou rectangulaires avec des côtés parallèles aux axes. Les méthodes de partitionnement et de recherche dans ce type de baquets sont alors relativement simples elles aussi ;
- des baquets à forme plus complexe, permettant de garantir un nombre constant d'objets dans chacun des baquets par exemple. Cette dernière catégorie est particulièrement bien adaptée à la recherche de solutions globales, où chaque objet de la région initiale doit être considéré.

Dans notre contexte, la détection de ligne tiretées (chapitre 6), cette structure de baquet n'a cependant pas besoin d'être aussi sophistiquée, son utilisation se limitant à des recherches locales. Nous avons donc opté pour une définition de baquets à forme simple, en y intégrant une des contraintes de la détection de lignes tiretées : la distance maximum autorisée entre deux segments supportés par la même ligne tiretées. Cette distance est utilisée pour fixer les dimensions des baquets ; en considérant l'extrémité d'une clé c appartenant à une hypothèse h, on est ainsi assuré que les clés pouvant étendre h ont au moins une extrémité dans le baquet auquel appartient c ou l'un de ses baquets voisins. La recherche des clés se trouvant à proximité du point p se décompose alors de la manière suivante :

1. détermination du baquet B où se trouve le point p,
2. détermination des clés se trouvant dans le baquet B et dans les baquets qui lui sont voisins.

Bibliographie

[Abe 86] K. Abe, Y. Azumatani, M. Mukouda et S. Suzuki. Discrimination of Symbols, Lines, and Characters in Flowchart Recognition. *Proceedings of 8th International Conference on Pattern Recognition, Paris (France)*, pages 1071–1074, 1986. (p. 8)

[Ada 00] S. Adam, R. Mullot, J.-M. Ogier, C. Cariou, J. Gardes et Y. Lecourtier. Interprétation des documents du réseau téléphonique : approche multi-spécialistes. *Actes du 12e Congrès de Reconnaissance des Formes et Intelligence Artificielle (RFIA'2000), Paris*, volume 3, pages 357–364, février 2000. (p. 8)

[Aga 96] G. Agam, H. Luo et I. Dinstein. Morphological Approach for Dashed Lines Detection. In Kasturi and Tombre [Kas 96], pages 92–105. (p. 26)

[Ale 90] F. Alexandre. *Une modélisation fonctionnelle du cortex : la colonne corticale. Aspects visuels et moteurs*. Thèse de doctorat, Université de Nancy I, Vandœuvre-lès-Nancy, 1990. (p. 91)

[Anq 99] E. Anquetil, B. Coüasnon et F. Dambreville. A Symbol Classifier able to Reject Wrong Shapes for Document Recognition Systems. *Proceedings of 3rd International Workshop on Graphics Recognition, Jaipur (India)*, pages 195–202, septembre 1999. (pp. 8, 92)

[Ant 92] D. Antoine, S. Collin et K. Tombre. Analysis of Technical Documents: The REDRAW System. H. S. Baird, H. Bunke et K. Yamamoto, éditeurs, *Structured Document Image Analysis*, pages 385–402. Springer-Verlag, Berlin/Heidelberg, 1992. (p. 54)

[Aok 96] Y. Aoki, A. Shio, H. Arai et K. Odaka. A Prototype System for Interpreting Hand-Sketched Floor Plans. *Proceedings of the 13th International Conference on Pattern Recognition, Vienna (Austria)*, volume 3, pages 747–751, août 1996. (p. 10)

[Arc 85] C. Arcelli et G. Sanniti di Baja. A Width-Independent Fast Thinning Algorithm. *IEEE Transactions on PAMI*, 7(4):463–474, 1985. (p. 50)

[Arc 89] C. Arcelli et G. Sanniti di Baja. A One-Pass Two-Operation Process to Detect the Skeletal Pixels on the 4-Distance Transform. *IEEE Transactions on PAMI*, 11(4):411–414, 1989. (p. 50)

[Arm 93] J.-P. Armand. Musical Score Recognition: A Hierarchical and Recursive Approach. *Proceedings of 2nd International Conference on Document Analysis and Recognition, Tsukuba (Japan)*, pages 906–909, 1993. (p. 91)

[AS 97] C. Ah-Soon. Symbol Detection in Architectural Drawings. *Proceedings of 2nd International Workshop on Graphics Recognition, Nancy (France)*, pages 280–286, août 1997. (p. 108)

[AS 98a] C. Ah-Soon. A Constraint Network for Symbol Detection in Architectural Drawings. In Tombre and Chhabra [Tom 98c], pages 80–90. (p. 108)

[AS 98b] C. Ah-Soon. *Analyse de plans architecturaux*. Thèse de doctorat, Institut National Polytechnique de Lorraine, Vandœuvre-lès-Nancy, octobre 1998. (pp. 30, 54, 106, 109)

[AS 98c] C. Ah-Soon et Ph. Dosch. Une interface homme-machine pour l'analyse de plans architecturaux. *Actes du 1er Colloque International Francophone sur l'Écrit et le Document, Québec (Canada)*, pages 92–101, mai 1998. (p. 165)

[AS 98d] C. Ah-Soon et K. Tombre. Network-Based Recognition of Architectural Symbols. A. Amin, D. Dori, P. Pudil et H. Freeman, éditeurs, *Advances in Pattern Recognition (Proceedings of Joint IAPR Workshops SSPR'98 and SPR'98, Sydney, Australia)*, volume 1451, série *Lecture Notes in Computer Science*, pages 252–261, août 1998. (p. 108)

[Asa 85] T. Asano, M. Edahiro, H. Imai, M. Iri et K. Murota. Practical Use of Bucketing Techniques in Computational Geometry. G. T. Toussaint, éditeur, *Computational Geometry*, pages 153–195. North-Holland, 1985. (pp. 72, 189)

[Atu 99] A.S. Atukorale et P.N. Suganthan. Combining Classifiers Based on Confidence Values. *Proceedings of 5th International Conference on Document Analysis and Recognition, Bangalore (India)*, pages 37–40, septembre 1999. (p. 92)

[Bai 97] D. Bainbridge et N. Carter. Automatic Reading of Music Notation. In Bunke and Wang [Bun 97], chapitre 22, pages 583–603. (p. 8)

[Bal 82] D.H. Ballard et C.M. Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs (NJ), USA, 1982. (p. 139)

[Bap 98] F. Bapst. *Reconnaissance de documents assistée : architecture logicielle et intégration de savoir-faire*. Thèse de Doctorat, Faculté des sciences de l'université de Fribourg (Suisse), octobre 1998. (p. 165)

[Ber 73] A.T. Bertztiss. A Backtrack Procedure for Isomorphism of Directed Graphs. *Journal of the ACM*, 20(3):365–377, 1973. (p. 93)

[Bha 94] S. Bhattacharjee et G. Monagan. Recognition of Cartographic Symbols. *Proceedings of IAPR Workshop on Machine Vision Applications, Kawasaki (Japan)*, pages 226–229, 1994. (p. 100)

[Blo 92] D. Blostein et H. S. Baird. A Critical Survey of Music Image Analysis. H. S. Baird, H. Bunke et K. Yamamoto, éditeurs, *Structured Document Image Analysis*, pages 405–434. Springer-Verlag, Heidelberg, 1992. (p. 8)

[Blo 99] D. Blostein et L. Haken. Using Diagram Generation Software to Improve Diagram Recognition: A Case Study of Music Notation. *IEEE Transactions on PAMI*, 21(11):1121–1136, novembre 1999. (p. 8)

[Boa 92] L. Boatto, V. Consorti, M. Del Buono, S. Di Zenzo, V. Eramo, A. Esposito, F. Melcarne, M. Meucci, A. Morelli, M. Mosciatti, S. Scarci et M. Tucci. An Interpretation System for Land Register Maps. *IEEE COMPUTER Magazine*, 25(7):25–33, juillet 1992. (pp. 66, 68)

[Bol 83] R. C. Bolles et R. A. Cain. Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method. A. Pugh, éditeur, *Robot Vision*, pages 43–82. IFS Publications Ltd. (United Kingdom) and Springer-Verlag (Berlin), 1983. (pp. 96, 136, 138)

[Bor 86] G. Borgefors. Distance Transforms in Digital Images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986. (p. 50)

[Bot 95] P. Bottoni, U. Cugini, P. Mussio, C. Papetti et M. Protti. A System for Form-Feature Based Interpretation of Technical Drawings. *Machine Vision and Applications*, 8:326–335, 1995. (p. 8)

[Bun 92] H. Bunke. Recent Advances in String Matching. H. Bunke, éditeur, *Advances in Structural and Syntactic Pattern Recognition (Proceedings of International Workshop on Structural and Syntactic Pattern Recognition, Bern, Switzerland)*, volume 5, série *Series in Machine Perception and Artificial Intelligence*, pages 3–21. World Scientific, 1992. (p. 97)

[Bun 93a] H. Bunke. Structural and Syntactic Pattern Recognition. In Chen et al. [Che 93a], chapitre 1.5, pages 163–209. (p. 93)

[Bun 93b] H. Bunke et U. Bühler. Applications of Approximate String Matching to 2D Shape Recognition. *Pattern Recognition*, 26(12):1797–1812, 1993. (p. 117)

[Bun 95] H. Bunke et B. T. Messmer. Efficient Attributed Graph Matching and its Application to Image Analysis. C. Braccini, L. De Floriani et G. Vernazza, éditeurs, *Proceedings of 8th International Conference on Image Analysis and Processing, San Remo (Italy)*, volume 974, série *Lecture Notes in Computer Science*, pages 45–55, septembre 1995. (p. 105)

[Bun 97] H. Bunke et P. S. P. Wang, éditeurs. *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997. (pp. 192, 197, 198)

[Can 86] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on PAMI*, 8(6):679–698, 1986. (p. 23)

[Car 83] S.K. Card, T.P. Moran et A. Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983. (p. 148)

[Ces 95] F. Cesarini, M. Gori, S. Marinai et G. Soda. A Hybrid System for Locating and Recognizing Low Level Graphic Items. *Proceedings of 1st International Workshop on Graphics Recognition, The Penn State Scanticon, University Park, PA, (USA)*, pages 62–70, août 1995. (p. 92)

[Che 93a] C. H. Chen, L. F. Pau et P. S. P. Wang, éditeurs. *Handbook of Pattern Recognition and Computer Vision*. World Scientific, Singapore, 1993. (pp. 193, 196, 200)

[Che 93b] Y.-S. Chen et W.-H. Hsu. Parallel Thinning Algorithms for Binary Digital Patterns. In Chen et al. [Che 93a], chapitre 2.7, pages 457–490. (p. 50)

[Che 96] Y. Chen, N. A. Langrana et A. K. Das. Perfecting Vectorized Mechanical Drawings. *Computer Vision and Image Understanding*, 63(2):273–286, mars 1996. (p. 63)

[Che 99] H.D. Cheng et Y.-H. Chen. Fuzzy Partition of Two-Dimensional Histogram and its Application to Thresholding. *Pattern Recognition*, 32:825–843, 1999. (p. 24)

[Chh 98a] A. K. Chhabra. Graphic Symbol Recognition: An Overview. In Tombre and Chhabra [Tom 98c], pages 68–79. (p. 90)

[Chh 98b] A. K. Chhabra et I. T. Phillips. The Second International Graphics Recognition Contest—Raster to Vector Conversion: A Report. In Tombre and Chhabra [Tom 98c], pages 390–410. (p. 16)

[Chh 99] A. K. Chhabra et I. T. Phillips. Edit Cost as a Measure of Performance of Graphics Recognition Systems. *Proceedings of 3rd International Workshop on Graphics Recognition, Jaipur (India)*, pages 331–334, septembre 1999. (pp. 16, 178)

[Cho 65] N. Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1965. (p. 96)

[Clo 99] R. Clouard, A. Elmoataz et M. Revenu. Une méthodologie de développement d'applications de traitement d'images. *Actes du 17^e Colloque sur le Traitement du Signal et des Images (*GRETSI*), Vannes*, pages 323–326, septembre 1999. (p. 166)

[Col 92] S. Collin. *Interprétation de la cotation des dessins techniques par analyse syntaxique*. Thèse de doctorat, Institut National Polytechnique de Lorraine, Vandœuvre-lès-Nancy, janvier 1992. (pp. 8, 31, 97, 178)

[Con 88] R. S. Conker. A Dual Plane Variation of the Hough Transform for Detecting Non-concentric Circles of Different Radii. *Computer Vision, Graphics and Image Processing*, 43:115–132, 1988. (p. 78)

[Cou 90] J. Coutaz. *Interfaces homme-ordinateur*. Dunod Informatique, Paris, avril 1990. (pp. 147, 151, 154)

[Cro 96] A. D. J. Cross et E. R. Hancock. Inexact Graph Matching with Genetic Search. P. Perner, P. Wang et A. Rosenfeld, éditeurs, *Advances in Structural and Syntactial Pattern Recognition (Proceedings of 6th International SSPR Workshop, Leipzig, Germany)*, volume 1121, série *Lecture Notes in Computer Science*, pages 150–159. Springer-Verlag, août 1996. (pp. 99, 100)

[Dal 99] M.K. Dalheimer. *Programming with Qt*. O'Reilly, mai 1999. (pp. 154, 176, 188)

[dB 94] G. Sanniti di Baja. Well-Shaped, Stable, and Reversible Skeletons from the (3,4)-Distance Transform. *Journal of Visual Communication and Image Representation*, 5(1):107–115, 1994. (pp. 50, 51)

[dH 94] J. E. den Hartog et T. K. ten Kate. Finding Arrows in Utility Maps using a Neural Network. *Proceedings of the 12th International Conference on Pattern Recognition, Jerusalem (Israel)*, volume 2, pages 190–194, 1994. (p. 92)

[dH 96] J. E. den Hartog, T. K. ten Kate et J. J. Gerbrands. Knowledge-Based Interpretation of Utility Maps. *Computer Vision and Image Understanding*, 63(1):105–117, janvier 1996. (p. 8)

[dJ 95] E. Oliveira de Jesus. ECIR — An Electronic Circuit Images Recognizer. *Proceedings of 1st International Workshop on Graphics Recognition, The Penn State Scanticon, University Park, PA, (USA)*, pages 252–261, août 1995. (p. 8)

[Doe 96] D. Doermann, E. Rivlin et I. Weiss. Applying algebraic and differential invariants for logo recognition. *Machine Vision and Applications*, 9(2):73–86, 1996. (p. 90)

[Doe 98] David S. Doermann. An Introduction to Vectorization and Segmentation. In Tombre and Chhabra [Tom 98c], pages 1–8. (p. 26)

[Dor 88] D. Dori et A. Pnueli. The Grammar of Dimensions in Machine Drawings. *Computer Vision, Graphics and Image Processing*, 42:1–18, 1988. (p. 97)

[Dor 89] D. Dori. A Syntactic/Geometric Approach to Recognition of Dimensions in Engineering Drawings. *Computer Vision, Graphics and Image Processing*, 47:271–291, 1989. (pp. 8, 97)

[Dor 93] D. Dori, Y. Liang, J. Dowell et I. Chai. Sparse-Pixel Recognition of Primitives in Engineering Drawings. *Machine Vision and Applications*, 6:69–82, 1993. (p. 57)

[Dor 95a] D. Dori. Representing pattern recognition-embedded systems through object-process diagrams: the case of the machine drawing understanding system. *Pattern Recognition Letters*, 16:377–384, avril 1995. (pp. 146, 159, 165)

[Dor 95b] D. Dori. Vector-based Arc Segmentation in the Machine Drawing Understanding System Environment. A. L. Spitz et A. Dengel, éditeurs, *Document Analysis Systems*, pages 338–362. World Scientific, 1995. (pp. 77, 79)

[Dor 96a] D. Dori et L. Wenyin. Vector-Based Segmentation of Text Connected to Graphics in Engineering Drawings. P. Perner, P. Wang et A. Rosenfeld, éditeurs, *Advances in Structural and Syntactial Pattern Recognition (Proceedings of 6th International SSPR Workshop, Leipzig, Germany)*, volume 1121, série *Lecture Notes in Computer Science*, pages 322–331. Springer-Verlag, août 1996. (p. 27)

[Dor 96b] D. Dori, L. Wenyin et M. Peleg. How to Win a Dashed Line Detection Contest. In Kasturi and Tombre [Kas 96], pages 286–300. (pp. 67, 68, 69)

[Dor 98a] D. Dori et Y. Velkovitch. Segmentation and Recognition of Dimensioning Text from Engineering Drawings. *Computer Vision and Image Understanding*, 69(2):196–201, février 1998. (p. 8)

[Dor 98b] Dov Dori et Wenyin Liu. Stepwise recovery of arc segmentation in complex line environments. *International Journal on Document Analysis and Recognition*, 1(1):62–71, février 1998. (pp. 77, 80)

[Dor 99] D. Dori et W. Liu. Sparse Pixel Vectorization: An Algorithm and Its Performance Evaluation. *IEEE Transactions on PAMI*, 21(3):202–215, mars 1999. (p. 80)

[Dos 97] Ph. Dosch et G. Masini. Urban Environment Modelling by Fusion of a Cadastral Map and a Digital Elevation Model. *Proceedings of 10th Scandinavian Conference on Image Analysis, Lappeenranta (Finland)*, pages 431–437, juin 1997. (p. 8)

[Dos 99a] Ph. Dosch, C. Ah-Soon, G. Masini, G. Sánchez et K. Tombre. Design of an Integrated Environment for the Automated Analysis of Architectural Drawings. S.-W. Lee et Y. Nakano, éditeurs, *Document Analysis Systems: Theory and Practice. Selected papers from Third IAPR Workshop, DAS'98, Nagano, Japan, November 4–6, 1998, in revised version*, Lecture Notes in Computer Science 1655, pages 295–309. Springer-Verlag, Berlin, 1999. (pp. 156, 165)

[Dos 99b] Ph. Dosch et G. Masini. Reconstruction of the 3D Structure of a Building from the 2D Drawings of its Floors. *Proceedings of 5th International Conference on Document Analysis and Recognition, Bangalore (India)*, pages 487–490, septembre 1999. (p. 133)

[Dos 00a] Ph. Dosch et G. Masini. Techniques de mise en correspondance en analyse de plans d'architecte. *Actes du 12e Congrès de Reconnaissance des Formes et Intelligence Artificielle (RFIA'2000), Paris*, volume 1, pages 453–462, février 2000. (pp. 34, 133)

[Dos 00b] Ph. Dosch, G. Masini et K. Tombre. Improving Arc Detection in Graphics Recognition. *Proceedings of the 15th International Conference on Pattern Recognition, Barcelona (Spain)*, volume 2, pages 243–246, septembre 2000. (p. 81)

[Dos 00c] Ph. Dosch, K. Tombre, Ch. Ah-Soon et G. Masini. A Complete System for the Analysis of Architectural Drawings. *International Journal on Document Analysis and Recognition*, septembre 2000. À paraître. (p. 165)

[Fah 88] C. S. Fahn, J. F. Wang et J. Y. Lee. A Topology-Based Component Extractor for Understanding Electronic Circuit Diagrams. *Computer Vision, Graphics and Image Processing*, 44:119–138, 1988. (p. 8)

[Fah 92] H. Fahmy et D. Blostein. A Survey of Graph Grammars: Theory and Applications. *Proceedings of 11th International Conference on Pattern Recognition, Den Haag (The Netherlands)*, volume 2, pages 294–298, 1992. (p. 97)

[Fau 81] O. D. Faugeras et M. Berthod. Improving Consistency and Reducing Ambiguity in Stochastic Labeling: An Optimization Approach. *IEEE Transactions on PAMI*, 3:412–423, 1981. (p. 95)

[Fed 71] J. Feder. Plex Languages. *Information Science*, 3:225–241, 1971. (p. 97)

[Fil 95] A. Filbois. *Contributions à la modélisation automatique d'objets polyédriques 3D : extraction des primitives 3D, facettes et segments*. Thèse de doctorat, Institut National Polytechnique de Lorraine, Vandœuvre-lès-Nancy, juillet 1995. (p. 59)

[Fle 88] L. A. Fletcher et R. Kasturi. A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images. *IEEE Transactions on PAMI*, 10(6):910–918, 1988. (pp. 26, 30)

[Fre 75] H. Freeman et R. Shapira. Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve. *Communications of the ACM*, 18(7):409–413, juillet 1975. (p. 27)

[Fu 74] K. S. Fu. *Syntactic Methods in Pattern Recognition*, volume 112, série *Mathematics in Science and Engineering*. Academic Press, New York, 1974. (pp. 93, 96)

[Fu 83] K. S. Fu. A Step Towards Unification of Syntactic and Statistical Pattern Recognition. *IEEE Transactions on PAMI*, 5(2):200–205, 1983. (pp. 100, 116)

[Fuk 93] K. Fukunaga. Statistical Pattern Recognition. In Chen et al. [Che 93a], chapitre 1.2, pages 33–60. (p. 90)

[Fut 90] T. Futatsumata, G. Shichino, J. Shibayama et A. Maeda. Development of an Automatic Recognition System for Plant Diagrams. *Proceedings of IAPR Workshop on Machine Vision Applications, Tokyo (Japan)*, pages 207–210, 1990. (p. 8)

[Gar 79] M.R. Garey et D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and company, 1979. (pp. 93, 96, 138)

[Gar 00] D. Garlan et M. Shaw. An Introduction to Software Architecture. CMU-CS-94-166. `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/able/ftp/intro_softarch.ps`, avril 2000. (p. 152)

[Gau 96] M.-C. Gaudel, B. Marre, F. Schlienger et G. Bernot. *Précis de génie logiciel*. Masson, Paris, mai 1996. (p. 152)

[Gér 99] T. Géraud, Y. Fabre, D. Papadopoulos-Orfanos et J.F. Mangin. Vers une réutilisabilité totale des algorithmes de traitement d'images. *Actes du 17^e Colloque sur le Traitement du Signal et des Images (*GRETSI*), Vannes*, pages 331–334, septembre 1999. (p. 156)

[Gol 89] D.E Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989. (p. 98)

[Gon 87] R. C. Gonzalez et P. Wintz. *Digital Image Processing*. Addison-Wesley, 1987. (p. 97)

[Goo 85] L. Van Gool, P. Dewaele et A. Oosterlinck. Texture Analysis Anno 1983. *Computer Vision, Graphics and Image Processing*, 29:336–357, 1985. (p. 115)

[Gou 00] V. Gough. LIMP, The Large Image Manipulation Program. `http://www.remotesensing.org/docs/limp-ht/`, avril 2000. (p. 34)

[Gra 85] C. Granger. Reconnaissance d'objets par mise en correspondance en vision par ordinateur. Thèse de doctorat, Université de Nice, 1985. (p. 94)

[Gra 00] Silicon Graphics. ImageVision Library Programming Guide. `http://shiva.missouri.edu:88/SGI_Developer/IL_PG/3/`, avril 2000. (p. 159)

[Gro 95] M. D. Gross. Indexing Visual Databases of Designs with Diagrams. A. Koutamanis, H. Timmermans et I. Vermeulen, éditeurs, *Visual Databases in Architecture*, chapitre 1, pages 1–14. Avebury, 1995. (p. 10)

[Hab 93] A. H. Habacha. *Reconnaissance de symboles techniques et analyse contextuelle de schémas*. Thèse de doctorat, Institut National Polytechnique de Lorraine, Vandœuvre-lès-Nancy, juin 1993. (p. 95)

[Han 94] C.-C. Han et K.-C. Fahn. Skeleton Generation of Engineering Drawings via Contour Matching. *Pattern Recognition*, 27(2):261–275, 1994. (p. 54)

[Har 92a] R. M. Haralick. Performance Characterization in Image Analysis: Thinning, a Case in Point. *Pattern Recognition Letters*, 13(1):5–12, 1992. (p. 15)

[Har 92b] R. M. Haralick et L. G. Shapiro. *Machine Vision*. Addison-Wesley, Reading, Massachusetts, 1992. (p. 78)

[Hor 89] R. Horaud et T. Skordas. Stereo Correspondance Through Feature Grouping and Maximal Cliques. *IEEE Transactions on PAMI*, 11(11):1168–1180, 1989. (p. 138)

[Hor 92] O. Hori et A. Okazaki. High Quality Vectorization Based on a Generic Object Model. H. S. Baird, H. Bunke et K. Yamamoto, éditeurs, *Structured Document Image Analysis*, pages 325–339. Springer-Verlag, Heidelberg, 1992. (p. 58)

[Hut 93] D. P. Huttenlocher et W. J. Rucklidge. A Multi-Resolution Technique for Comparing Images Using the Hausdorff Distance. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, New York, NY (USA)*, pages 705–706, 1993. (p. 39)

[Ill 88] J. Illingworth et J. Kittler. A Survey of the Hough Transform. *Computer Vision, Graphics and Image Processing*, 44:87–116, 1988. (p. 78)

[Jai 93] M. Y. Jaisimha, R. M. Haralick et D. Dori. A Methodology for the Characterization of the Performance of Thinning Algorithms. *Proceedings of 2nd International Conference on Document Analysis and Recognition, Tsukuba (Japan)*, pages 282–286, 1993. (p. 16)

[Jan 94] R. D. T. Janssen et A. M. Vossepoel. Compilation of mosaics from separately scanned line drawings. *Proceedings of the 2nd IEEE Workshop on Applications of Computer Vision*, pages 36–43, Sarasota, Florida, USA, décembre 1994. (p. 34)

[Jan 97a] R. D. T. Janssen. Interpretation of Maps: From Bottom-Up to Model-Based. In Bunke and Wang [Bun 97], chapitre 20, pages 529–555. (p. 8)

[Jan 97b] R. D. T. Janssen et A. M. Vossepoel. Adaptive Vectorization of Line Drawing Images. *Computer Vision and Image Understanding*, 65(1):38–56, janvier 1997. (pp. 63, 178)

[Jia 93] X. Y. Jiang et H. Bunke. An Optimal Algorithm for Extracting the Regions of a Plane Graph. *Pattern Recognition Letters*, 14:553–558, 1993. (pp. 116, 118, 135)

[Jia 99] X. Jiang, A. Münger et H. Bunke. Synthesis of Representative Graphical Symbols by Computing Generalized Median Graph. *Proceedings of 3rd International Workshop on Graphics Recognition, Jaipur (India)*, pages 187–194, septembre 1999. (pp. 99, 112)

[Jon 99] A. Jonk, R. Van Den Boomgaard et A. Smeulders. Grammatical Inference of Dashed Lines. *Computer Vision and Image Understanding*, 74(3):212–226, 1999. (pp. xiv, 67, 68)

[Jos 92] S. H. Joseph et T. P. Pridmore. Knowledge-Directed Interpretation of Mechanical Engineering Drawings. *IEEE Transactions on PAMI*, 14(9):928–940, septembre 1992. (pp. 67, 68)

[Kas 90] R. Kasturi, S. T. Bow, W. El-Masri, J. Shah, J. R. Gattiker et U. B. Mokate. A System for Interpretation of Line Drawings. *IEEE Transactions on PAMI*, 12(10):978–992, 1990. (pp. 66, 68)

[Kas 96] R. Kasturi et K. Tombre, éditeurs. *Graphics Recognition—Methods and Applications*, volume 1072, série *Lecture Notes in Computer Science*. Springer-Verlag, mai 1996. (pp. 191, 194, 197, 199, 200, 201)

[Kit 87] J. Kittler. Statistical Pattern Recognition: The State of the Art. V. Cantoni, V. Di Gesù et S. Levialdi, éditeurs, *Image Analysis and Processing*, volume 2, pages 57–66. Plenum Press, 1987. (p. 90)

[Knu 68] D. E. Knuth. Semantics of context-free languages. *Journal of Mathematic and System Theory*, 2:127–146, 1968. (p. 96)

[Koh 94] C. Kohl et J. Mundy. The Development of the Image Understanding Environment. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Seattle, WA (USA)*, pages 443–447, 1994. (pp. 145, 159)

[Kol 97] C. Kolski. *Interfaces homme-machine*. Hermès, Paris, 2 édition, janvier 1997. (pp. 147, 152, 154)

[Kon 96] B. Kong, I. T. Phillips, R. M. Haralick, A. Prasad et R. Kasturi. A Benchmark: Performance Evaluation of Dashed-Line Detection Algorithms. In Kasturi and Tombre [Kas 96], pages 270–285. (p. 16)

[Kou 90] A. Koutamanis. *Development of a Computerized Handbook of Architectural Plans*. Ph.D. Thesis, Technische Universiteit Delft, Delft, Netherlands, septembre 1990. (p. 9)

[Kou 95] A. Koutamanis. Recognition and Retrieval in Visual Architectural Databases. A. Koutamanis, H. Timmermans et I. Vermeulen, éditeurs, *Visual Databases in Architecture*, chapitre 2, pages 15–42. Avebury, 1995. (p. 9)

[Lai 91] C. P. Lai et R. Kasturi. Detection of Dashed Lines in Engineering Drawings and Maps. *Proceedings of 1st International Conference on Document Analysis and Recognition, Saint-Malo, France*, volume 2, pages 507–515, 1991. (pp. 66, 68)

[Lam 92] L. Lam, S.-W. Lee et C. Y. Suen. Thinning Methodologies — A Comprehensive Survey. *IEEE Transactions on PAMI*, 14(9):869–885, septembre 1992. (p. 49)

[Lam 93] L. Lam et C. Y. Suen. Evaluation of Thinning Algorithms from an OCR Viewpoint. *Proceedings of 2nd International Conference on Document Analysis and Recognition, Tsukuba (Japan)*, pages 287–290, 1993. (p. 49)

[Lam 94] S. W. C. Lam et H. H. C. Ip. Structural Texture Segmentation Using Irregular Pyramid. *Pattern Recognition Letters*, pages 691–698, juillet 1994. (pp. 116, 119)

[Lam 97] L. Lam, Y.-S. Huang et C. Y. Suen. Combination of Multiple Classifier Decisions for Optical Character Recognition. In Bunke and Wang [Bun 97], chapitre 3, pages 79–101. (p. 92)

[Lee 92] S.-W. Lee. Recognizing Hand-Drawn Electrical Circuit Symbols with Attributed Graph Matching. H. S. Baird, H. Bunke et K. Yamamoto, éditeurs, *Structured Document Image Analysis*, pages 340–358. Springer-Verlag, Heidelberg, 1992. (pp. 8, 94)

[Lin 85] X. Lin, S. Shimotsuji, M. Minoh et T. Sakai. Efficient Diagram Understanding with Characteristic Pattern Detection. *Computer Vision, Graphics and Image Processing*, 30:84–106, 1985. (pp. xiii, 57, 91)

[Lin 97] S.-C. Lin et C.-K. Ting. A new approach for detection of dimensions set in mechanical drawings. *Pattern Recognition Letters*, 18(4):367–373, avril 1997. (p. 8)

[Lla 97a] J. Lladós. *Combining Graph Matching and Hough Transform for Hand-Drawn Graphical Document Analysis. Application to Architectural Drawings.* Thèse de doctorat, Universitat Autònoma de Barcelona (Espagne)/Université Paris 8 (France), septembre 1997. (p. 94)

[Lla 97b] J. Lladós, J. López-Krahe et E. Martí. A System to Understand Hand-Drawn Floor Plans Using Subgraph Isomorphism and Hough Transform. *Machine Vision and Applications*, 10(3):150–158, 1997. (p. 10)

[Lla 98] J. Lladós, G. Sánchez et E. Martí. A String Based Method to Recognize Symbols and Structural Textures in Architectural Plans. In Tombre and Chhabra [Tom 98c], pages 91–103. (pp. 10, 116)

[Lla 99] J. Lladós et E. Martí. A Graph-Edit Algorithm for Hand-Drawn Graphical Document Recognition and Their Automatic Introduction into CAD Systems. *Machine Graphics & Vision*, 8(2):195–211, 1999. (p. 8)

[Lla 00] J. Lladós, J. López-Krahe, G. Sánchez et E. Martí. Interprétation de cartes et plans par mise en correspondance de graphes d'attributs. *Actes du 12e Congrès de Reconnaissance des Formes et Intelligence Artificielle (RFIA'2000), Paris*, volume 3, pages 225–234, février 2000. (p. 10)

[Low 87] D. Lowe. Three-Dimensional Object Recognition from Single Two-Dimensional Images. *Artificial Intelligence*, 31:355–395, 1987. (pp. 60, 78)

[Luo 94] H. Luo et I. Dinstein. Using Directional Mathematical Morphology for Separation of Character Strings from Text/Graphics Image. *Shape, Structure and Pattern Recognition (Post-proceedings of IAPR Workshop on Syntactic and Structural Pattern Recognition, Nahariya, Israel)*, pages 372–381. World Scientific, 1994. (p. 27)

[Mae 91] M. Maes. Polygonal Shape Recognition Using String-Matching Techniques. *Pattern Recognition*, 24(5):433–440, 1991. (p. 117)

[Mar 89] P. Martin. Reconnaissance de Partitions Musicales et Réseaux de Neurones : une Etude. *Actes du 7e Congrès AFCET de Reconnaissance des Formes et Intelligence Artificielle, Paris*, pages 217–226, 1989. (p. 92)

[Mar 91] P. Martin et C. Bellissant. Low-Level Analysis of Music Drawing Images. *Proceedings of 1st International Conference on Document Analysis and Recognition, Saint-Malo, France*, volume 1, pages 417–425, 1991. (p. 92)

[Mar 97] R. Mariani, M.P. Deseilligny, J. Labiche et R. Mullot. Linear Texture Segmentation Using Elastic Model Matching. Application for Geographic Map Understanding. *Proceedings of 2nd International Workshop on Graphics Recognition, Nancy (France)*, pages 201–208, 1997. (p. 67)

[Mas 83] G. Masini et R. Mohr. MIRABELLE: A System for Structural Analysis of Drawings. *Pattern Recognition*, 16(4):363–372, 1983. (p. 96)

[Mat 00] P. Mattis et S. Kimball. GIMP: The GNU Image Manipulation Program. `http://www.gimp.org/`, avril 2000. (p. 172)

[Meh 99] K. Mehlhorn et S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, England, 1999. (p. 156)

[Meh 00] K. Mehlhorn, S. Näher et C. Uhrig. LEDA, Library of Efficient Data Types and Algorithms. `http://www.mpi-sb.mpg.de/LEDA/`, avril 2000. (p. 156)

[Mes 93] B. T. Messmer et H. Bunke. A Network Based Approach to Exact and Inexact Graph Matching. Technical Report IAM-93-021, Institut für Informatik und angewandte Mathematik, Universität Bern, septembre 1993. (pp. 105, 110)

[Mes 96] B. T. Messmer et H. Bunke. Automatic Learning and Recognition of Graphical Symbols in Engineering Drawings. In Kasturi and Tombre [Kas 96], pages 123–134. (p. 105)

[Mey 91] B. Meyer. *Conception et programmation par objets, Seconde édition*. Informatique Intelligence Artificielle. InterÉditions, Paris, 1991. (p. 153)

[Miy 96] H. Miyao et Y. Nakano. Note Symbol Extraction for Printed Piano Scores Using Neural Networks. *IEICE Transactions on Information and Systems*, E79-D(5):548–554, mai 1996. (p. 92)

[Moh 86] R. Mohr et T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28:225–233, 1986. (p. 95)

[Moh 88] R. Mohr et G. Masini. Good Old Discrete Relaxation. *Proceedings of 8th European Conference on Artificial Intelligence, Munich (West Germany)*, pages 651–656, 1988. (pp. 40, 95)

[Mon 91] A. Montanvert, P. Meer et A. Rosenfeld. Hierarchical Image Analysis using Irregular Tessellations. *IEEE Transactions on PAMI*, 13(4):307–316, 1991. (p. 116)

[Mun 92] J. Mundy, T. Binford, T. Boult, A. Hanson, R. Beveridge, R. Haralick, V. Ramesh, C. Kohl, D. Lawton, D. Morgan, K. Price et T. Strat. The Image Understanding Environment Program. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Urbana Champaign, IL (USA)*, pages 406–416, 1992. (p. 145)

[Mus 96] D. R. Musser et A. Saini. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. Addison-Wesley, Reading, MA, 1996. (p. 156)

[Nil 80] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, Palo Alto, California, 1980. (p. 94)

[O'G 95] L. O'Gorman et R. Kasturi. *Document Image Analysis*. IEEE Computer Society Press, 1995. (p. 22)

[Ogi 97] J.-M. Ogier, R. Mullot, J. Labiche et Y. Lecourtier. An Image Interpretation Device Can Not Be Reliable Without Any Semantic Coherency Analysis of the Interpretated Objects — Application to French Cadastral Maps. *Proceedings of 4th International Conference on Document Analysis and Recognition, Ulm (Germany)*, pages 532–535, août 1997. (p. 8)

[Oli 00] P. Oliver. Wotsit's Format. `http://www.wotsit.org/`, avril 2000. (p. 156)

[Pao 93] Y.-H. Pao. Neural Net Computing for Pattern Recognition. In Chen et al. [Che 93a], chapitre 1.4, pages 125–162. (p. 92)

[Pas 94] B. Pasternak. Processing Imprecise and Structural Distorted Line Drawings by an Adaptable Drawing Interpretation Kernel. *Proceedings of 1st IAPR Workshop on Document Analysis Systems, Kaiserslautern (Germany)*, pages 349–365, 1994. (p. 105)

[Pas 96] B. Pasternak. *Adaptierbares Kernsystem zur Interpretation von Zeichnungen.* Dissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.), Universität Hamburg, avril 1996. (p. 105)

[Phi 93] I. T. Phillips, S. Chen et R. M. Haralick. CD-ROM Document Database Standard. *Proceedings of 2nd International Conference on Document Analysis and Recognition, Tsukuba (Japan)*, pages 478–483, 1993. (p. 16)

[Phi 98] I. T. Phillips, J. Liang, A. K. Chhabra et Robert Haralick. A Performance Evaluation Protocol for Graphics Recognition Systems. In Tombre and Chhabra [Tom 98c], pages 372–389. (p. 16)

[Pic 00] M. Pichler. VRWEB: A Multi-System VRML Viewer. `http://www2.iicm.edu/vrweb/`, avril 2000. (p. 166)

[Pra 78] W. K. Pratt. *Digital Image Processing.* Wiley-Interscience, 1978. (p. 30)

[Pri 96] G. Priestnall, R. E. Marston et D. G. Elliman. Arrowhead recognition during automated data capture. *Pattern Recognition Letters*, 17(3):277–286, mars 1996. (p. 8)

[Ram 96] J.-Y. Ramel. *Interprétation automatique de dessins — méthodes d'analyse et de reconnaissance — application aux plans cinématiques.* Thèse de doctorat, Institut National des Sciences Appliquées, Lyon, novembre 1996. (p. 8)

[Ran 94] S. Randriamasy et L. Vincent. A Region-Based System for the Automatic Evaluation of Page Segmentation Algorithms. *Proceedings of 1st IAPR Workshop on Document Analysis Systems, Kaiserslautern (Germany)*, pages 29–44, 1994. (p. 16)

[Röö 96] M. Röösli et G. Monagan. Adding Geometric Constraints to the Vectorization of Line Drawings. In Kasturi and Tombre [Kas 96], pages 49–56. (pp. 63, 178)

[Ros 89] P. L. Rosin et G. A. West. Segmentation of Edges into Lines and Arcs. *Image and Vision Computing*, 7(2):109–114, mai 1989. (pp. 60, 78)

[Ros 97] P. L. Rosin. Techniques for Assessing Polygonal Approximation of Curves. *IEEE Transactions on PAMI*, 19(6):659–666, juin 1997. (pp. 62, 79, 85)

[Rou 98] M. Roux et H. Maître. Map Analysis for Guided Interpretation of Aerial Images. In Tombre and Chhabra [Tom 98c], pages 243–256. (p. 8)

[Ruc 97] W. J. Rucklidge. Efficiently Locating Objects Using the Hausdorff Distance. *International Journal of Computer Vision*, 24(3):251–270, 1997. (p. 39)

[Rya 95] K. Ryall et S. Shieber. Semi-Automatic Delineation of Regions on Floor Plans. *Proceedings of 3rd International Conference on Document Analysis and Recognition, Montréal (Canada)*, pages 964–969, août 1995. (pp. 9, 11)

[Sam 98] H. Samet et A. Soffer. MAGELLAN: Map Acquisition of GEographic Labels by Legend ANalysis. *International Journal on Document Analysis and Recognition*, 1(2):89–101, juin 1998. (pp. 8, 91)

[Sán 97] G. Sánchez, J. Lladós et E. Martí. Segmentation and Analysis of Linial Texture in Planes. *Proceedings of 7th Spanish National Symposium on Pattern Recognition and Image Analysis, Barcelona (Spain)*, volume 1, pages 401–406, 1997. (pp. 10, 116)

[Ser 82] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982. (p. 29)

[Sha 69] A. C. Shaw. A Formal Picture Description Scheme as a Basis for Picture Processing Systems. *Information and Control*, 14(1):9–52, 1969. (p. 96)

[Sha 81] L. G. Shapiro et R. Haralick. Structural Description and Inexact Matching. *IEEE Transactions on PAMI*, 3(5):504–519, 1981. (p. 94)

[Shn 98] B. Shneiderman. *Designing the User Interface*. Addison-Wesley, Reading, MA, 3 édition, mars 1998. (pp. 146, 151)

[Ski 98] S.S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, New York, 1998. (p. 156)

[Sme 96] A. W. M. Smeulders et T. ten Kate. Software System Design for Paper Map Conversion. In Kasturi and Tombre [Kas 96], pages 204–211. (p. 8)

[Sme 98] Arnold W. M. Smeulders et Carlo de Boer. Design and Performance in Object Recognition. In Tombre and Chhabra [Tom 98c], pages 335–346. (pp. 14, 16)

[Sof 98] A. Soffer et H. Samet. Using Negative Shape Features for Logo Similarity Matching. *Proceedings of the 14th International Conference on Pattern Recognition, Brisbane (Australia)*, pages 571–573, août 1998. (p. 91)

[Sue 92] P. Suetens, P. Fua et A. J. Hanson. Computational Strategies for Object Recognition. *ACM Computing Surveys*, 24(1):5–61, mars 1992. (p. 94)

[Tab 94] S. Tabbone. *Détection multi-échelle de contours subpixel et de jonctions*. Thèse de doctorat, Institut National Polytechnique de Lorraine, février 1994. (p. 54)

[Tab 00a] S. Tabbone et L. Wendling. Décomposition graphique sous forme de primitives 2D. *Actes du 2ème Colloque International Francophone sur l'Écrit et le Document, Lyon (France)*, pages 131–140, juillet 2000. (p. 63)

[Tab 00b] S. Tabbone et L. Wendling. Une méthode de binarisation par seuillage automatique. *Actes du 2ème Colloque International Francophone sur l'Écrit et le Document, Lyon (France)*, pages 333–341, juillet 2000. (p. 24)

[Tof 98] P. M. Tofani et R. Kasturi. Segmentation of Text from Color Map Images. Technical Report CSE-98-007, Departement of Computer Science and Engineering, mai 1998. (p. 27)

[Tom 87] K. Tombre. *La saisie automatisée de documents composites : reconnaissance, codage et interprétation des parties graphiques*. Thèse de doctorat, Institut National Polytechnique de Lorraine, Vandœuvre-lès-Nancy, 1987. (p. 59)

[Tom 95] K. Tombre et J.-C. Paul. Document Analysis: A Way to Integrate Existing Paper Information in Architectural Databases. A. Koutamanis, H. Timmermans et I. Vermeulen, éditeurs, *Visual Databases in Architecture*, chapitre 3, pages 43–52. Avebury, 1995. (p. 9)

[Tom 96a] K. Tombre. *Quelques contributions à l'interprétation de documents techniques*. Habilitation à diriger des recherches, Université Henri Poincaré Nancy I, février 1996. (p. 8)

[Tom 96b] K. Tombre. Structural and Syntactic Methods in Line Drawing Analysis: To Which Extent Do They Work? P. Perner, P. Wang et A. Rosenfeld, éditeurs, *Advances in Structural and Syntactial Pattern Recognition (Proceedings of 6th International SSPR Workshop, Leipzig, Germany)*, volume 1121, série *Lecture Notes in Computer Science*, pages 310–321. Springer-Verlag, août 1996. (p. 93)

[Tom 98a] K. Tombre, C. Ah-Soon, Ph. Dosch, A. Habed et G. Masini. Stable, Robust and Off-the-Shelf Methods for Graphics Recognition. *Proceedings of the 14th International Conference on Pattern Recognition, Brisbane (Australia)*, pages 406–408, août 1998. (pp. 47, 146)

[Tom 98b] K. Tombre et A. K. Chhabra. General Conclusions. In *Graphics Recognition—Algorithms and Systems* [Tom 98c], pages 411–420. (pp. 13, 26)

[Tom 98c] K. Tombre et A. K. Chhabra, éditeurs. *Graphics Recognition—Algorithms and Systems*, volume 1389, série *Lecture Notes in Computer Science*. Springer-Verlag, avril 1998. (pp. 191, 193, 194, 198, 200, 201, 202, 203)

[Tom 99] K. Tombre, C. Ah-Soon, Ph. Dosch, G. Masini et S. Tabbone. Stable and Robust Vectorization: How to Make the Right Choices. *Proceedings of 3rd International Workshop on Graphics Recognition, Jaipur (India)*, pages 3–16, septembre 1999. Revised version to appear in a forthcoming LNCS volume. (pp. 47, 48, 54, 63)

[Tom 00] K. Tombre et S. Tabbone. Vectorization in Graphics Recognition: To Thin or not to Thin. *Proceedings of the 15th International Conference on Pattern Recognition, Barcelona (Spain)*, volume 2, pages 91–96, septembre 2000. (p. 63)

[Tri 95a] Ø. Due Trier et A. K. Jain. Goal-Directed Evaluation of Binarization Methods. *IEEE Transactions on PAMI*, 17(12):1191–1201, décembre 1995. (p. 22)

[Tri 95b] Ø. Due Trier et T. Taxt. Evaluation of Binarization Methods for Document Images. *IEEE Transactions on PAMI*, 17(3):312–315, mars 1995. (pp. 16, 22)

[Tri 95c] Ø. Due Trier et T. Taxt. Improvement of "Integrated Function Algorithm" for Binarization of Document Images. *Pattern Recognition Letters*, 16:277–283, mars 1995. (p. 23)

[Tsa 80] W. H. Tsai et K. S. Fu. Attributed Grammar: A Tool for Combining Syntactic and Statistical Approaches to Pattern Recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(12):873–885, 1980. (pp. 97, 100)

[Tsa 84] W. H. Tsai et S. S. Yu. Attributed String Matching with Merging for Shape Recognition. *Proceedings of 7th International Conference on Pattern Recognition, Montréal (Canada)*, pages 1162–1164, 1984. (pp. 97, 117)

[Tsa 90] W. H. Tsai. Combining Statistical and Structural Methods. H. Bunke et A. Sanfeliu, éditeurs, *Syntactic and Structural Pattern Recognition: Theory and Applications*, chapitre 12, pages 349–366. World Scientific, 1990. (p. 100)

[Tüc 90] M. Tüceryan et A. K. Jain. Texture Segmentation Using Voronoi Polygons. *IEEE Transactions on PAMI*, 12(2):211–216, 1990. (p. 116)

[Ull 76] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *Journal of the ACM*, 23(1):31–42, 1976. (p. 94)

[Uns 86] M. Unser. Sum and Difference Histograms for Texture Classification. *IEEE Transactions on PAMI*, 8(1):118–125, 1986. (p. 116)

[Val 99] E. Valveny et E. Martí. Application of Deformable Template Matching to Symbol Recognition in Hand-written Architectural Drawings. *Proceedings of 5th International Conference on Document Analysis and Recognition, Bangalore (India)*, pages 483–486, septembre 1999. (p. 10)

[Vax 95] P. Vaxivière. *Interprétation de dessins techniques mécaniques*. Thèse de doctorat, Institut National Polytechnique de Lorraine, Vandœuvre-lès-Nancy, février 1995. (pp. 8, 57, 67, 68)

[Vos 97] A. M. Vossepoel, K. Schutte et C. F. P. Delanghe. Memory Efficient Skeletonization of Utility Maps. *Proceedings of 4th International Conference on Document Analysis and Recognition, Ulm (Germany)*, pages 797–800, août 1997. (pp. 34, 36)

[Wag 74] R. A. Wagner et M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974. (p. 97)

[Wal 84] K. Wall et P. Danielsson. A Fast Sequential Method for Polygonal Approximation of Digitized Curves. *Computer Vision, Graphics and Image Processing*, 28:220–227, 1984. (p. 59)

[Wan 97] Y.-K. Wang, K.-C. Fan et J.-T. Horng. Genetic-based search for error-correcting graph isomorphism. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 27(4):588–597, avril 1997. (p. 99)

[Wen 96] L. Wenyin et D. Dori. Sparse Pixel Tracking: A Fast Vectorization Algorithm Applied to Engineering Drawings. *Proceedings of the 13th International Conference on Pattern Recognition, Vienna (Austria)*, volume 3, pages 808–812, août 1996. (p. 57)

[Wen 98a] L. Wenyin et D. Dori. A Proposed Scheme for Performance Evaluation of Graphics/Text Separation Algorithms. In Tombre and Chhabra [Tom 98c], pages 359–371. (p. 16)

[Wen 98b] L. Wenyin et D. Dori. A Survey of Non-Thinning Based Vectorization Methods. A. Amin, D. Dori, P. Pudil et H. Freeman, éditeurs, *Advances in Pattern Recognition (Proceedings of Joint IAPR Workshops SSPR'98 and SPR'98, Sydney, Australia)*, volume 1451, série *Lecture Notes in Computer Science*, pages 230–241, août 1998. (p. 57)

[Wes 78] J. S. Weszka. A survey of threshold selection techniques. *Computer Graphics and Image Processing*, 7:259–265, 1978. (p. 22)

[Whi 83] J. M. White et G. D. Rohrer. Image Thresholding for Optical Character Recognition and Other Applications Requiring Character Image Extraction. *IBM Journal of Research and Development*, 27(4):400–411, 1983. (p. 23)

[Whi 98] A. P. Whichello et H. Yan. Document Image Mosaicing. *Proceedings of the 14th International Conference on Pattern Recognition, Brisbane (Australia)*, pages 1081–1083, août 1998. (p. 34)

[Wol 90] H. J. Wolfson. On Curve Matching. *IEEE Transactions on PAMI*, 12(5):483–489, 1990. (p. 97)

[Wun 00] R. Wunderling et M. Zöckler. DOC++, A Documentation System for C / C++ and Java. `http://www.zib.de/Visual/software/doc++/`, avril 2000. (p. 156)

[Zap 99] A. Zappalá, A. Gee et M. Taylor. Document Mosaicing. *Image and Vision Computing*, 17:589–595, 1999. (pp. 21, 34)

[Zen 96] S. Di Zenzo, L. Cinque et S. Levialdi. Run-Based Algorithms for Binary Image Analysis and Processing. *IEEE Transactions on PAMI*, 18(1):83–89, janvier 1996. (p. 57)

[Zes 94] S. Zesheng, J. Cunhong, L. Shifang et Y. Jing. Understanding Electronic Circuit Diagrams and Linking with ORCAD. *Proceedings of IAPR Workshop on Machine Vision Applications, Kawasaki (Japan)*, pages 243–246, 1994. (p. 8)

[Zhu 99] S.-C. Zhu. Stochastic Jump-Diffusion Process for Computing Medial Axes in Markov Random Fields. *IEEE Transactions on PAMI*, 21(11):1158–1169, novembre 1999. (p. 58)

Université HENRI POINCARÉ
NANCY 1

Monsieur **DOSCH Philippe**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 24 juillet 2000 n°391

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ, NANCY
LE PRÉSIDENT

G. BURLET

3C-F9

Université Henri Poincaré, Nancy-I
24-30 rue Lionnois - B.P. 3069 - 54013 NANCY CEDEX
Tél. : 03 83 85 48 00 - Fax : 03 83 85 48 48

Résumé

Cette thèse s'inscrit dans le domaine de l'analyse de documents, et porte plus précisément sur la reconnaissance de graphiques. Notre objectif est d'obtenir un modèle 3D d'un édifice à partir de ses plans d'architecte. Pour cela, nous avons intégré des modules d'analyse et une interface homme-machine (IHM) permettant à l'opérateur de contrôler les traitements à effectuer de manière optimale. La majeure partie de la thèse détaille les différents traitements mis en œuvre, du bas-niveau (sur des images de plans numérisés) jusqu'au haut-niveau (sur des données vectorisées). Nous décrivons les choix d'architecture logicielle qui nous ont menés à définir un système composé de trois couches hiérarchiques : une bibliothèque de composants logiciels, une couche applicative regroupant les différents outils de traitement d'images et de graphiques et l'IHM. Celle-ci permet d'interagir directement sur les données, de contrôler le déroulement de l'analyse et gère le dialogue homme-machine. Dans ce travail d'équipe, nos contributions principales portent sur l'organisation spatiale des traitements (tuilage), l'extraction d'indices de niveau intermédiaire (lignes tiretées, symboles tels que les cages d'escalier...), la mise en correspondance des étages pour la reconstruction 3D du bâtiment correspondant, l'intégration logicielle et la mise au point de tout l'aspect IHM.

Mots-clés: reconnaissance de graphiques, plans architecturaux, analyse de documents techniques, architecture logicielle, interface interactive, reconstruction 3D

Abstract

This thesis is in line with the field of document analysis, and more precisely deals with graphics recognition. Our purpose is the construction of a 3D model of a building from the architectural drawings of its floors. For that, we have a set of analysis modules and a graphical user interface (GUI) allowing a human operator to control the processings to be performed in an optimal way. The major part of this thesis describes the various processings implemented, from the low-level (bitmap images processings) to the high-level (vectorized data processings). We describe the choices which have led us to define a three-layered software architecture, hierarchally organized: A library of software components, an applicative layer grouping the various processings together and the GUI. The latter allows to directly interact on data to control the the analysis, and manages the man-machine cooperation. All the members of our research teams have been involved in this work, but our main contributions concern the design of the GUI, the spatial organization of processings (tiling), the extraction of middle-level features (dashed and dotted lines, symbols such as stairwell, etc.) and matching algorithms to construct the 3D structure of a building, as well as the software integration and the design of the GUI.

Keywords: graphics recognition, architectural drawings, technical documents analysis, software architecture, graphical user interface, 3D reconstruction