SOTISE: Un système ouvert pour le développement d'applications en traitement d'images

Christian Roberge, Carl Dionne* et Michael Schmidt
Technologies SEPIA,
Saint-Jean-sur-Richelieu, Qc, Canada J3B 6H9

Résumé

Cet article présente les différents concepts qui ont mené à l'élaboration d'un logiciel de traitement d'images. Le logiciel proposé permet de concilier les besoins de la recherche et ceux de l'industrie. Les caractéristiques particulières de son architecture permettent de vérifier et de valider facilement de nouvelles idées, tout en offrant la possibilité de les intégrer rapidement et efficacement à l'industrie. La formalisation de l'historique des traitements a également mené à un concept original, le graphe des manipulations. Par l'utilisation de techniques standard d'optimisation de graphe, le système possède la capacité de raisonner sur les traitements effectués et propose ainsi une solution à différents problèmes, dont la gestion de l'espace disque et le partage d'information dans une équipe de travail. L'article conclut par la présentation d'exemples d'applications qui illustreront les possibilités et la flexibilité du système SOTISE.

Abstract

This paper describes the different concepts that led to the creation of an image processing system. The proposed system allows us to meet the requirements of both research and production in industrial applications. Its unusual design not only allows SOTISE to easily test and verify new ideas, but also to integrate them rapidly and efficiently into production systems. The decision to keep a formal and automated record of image processing steps led to another original concept, that of the manipulation graph. Using standard graph manipulation techniques, this allows the system to manage and optimize the processing steps, and also provides a solution to other problems, such as managing disk storage and sharing information within a work group. This paper concludes by presenting sample applications which demonstrate the power and flexibility of the SOTISE system.

1 Introduction

Les contraintes spécifiques au traitement d'images en ont longtemps limité les applications industrielles. La puissance de calcul requise a d'abord conduit à des logiciels spécifiques fonctionnant sur un matériel spécialisé [1]. La recherche de solutions s'adaptant à un matériel en constante évolution a mené à la conception de librairies rudimentaires de traitement d'images, limitées aux algorithmes (Ex. Routines SPIDER de l'Université de Tokyo). L'utilisation de telles librairies impliquait toutefois un coût de développement élevé pour créer des applications spécifiques et des interfaces. Ce travail exigeait souvent une grande connaissance du matériel utilisé puisque de nombreux détails étaient laissés à l'utilisateur [2]. Au cours des dernières années, plusieurs logiciels complets et intégrés ont fait leur apparition (HIPS, HIPS-2[3], KBVision[1], Khoros [4], OBVIUS [5], WiT[2], etc.). Ces logiciels proposent un environnement de travail mieux adapté aux besoins des chercheurs.

Le foisonnement de nouvelles techniques (réseaux neuroniques, ondelettes et autres) et l'éclosion d'applications dans divers milieux (industriel, médical et autres) posent continuellement de nouveaux défis aux concepteurs de système de traitement d'images. Il est de plus en plus nécessaire de concilier les besoins de la recherche et développement et ceux des applications industrielles.

Le développement d'une application industrielle comporte deux phases distinctes: la recherche et développement, et la réalisation du produit final. Lors de la phase de développement, on vise l'application rapide d'un nouvel algorithme, l'intégration de plusieurs algorithmes et la possibilité de simuler

*e-mail: dionne@iro.umontreal.ca

[1] KBVision est un produit de Amerinex A.I., Inc.
[2] WiT est un produit de Logical Vision Ltd.

un environnement réel. Ce qui importe principalement, c'est la souplesse d'utilisation et la possibilité d'explorer rapidement de nouvelles voies.

Lors de la réalisation d'un produit industriel, il faut développer une interface spécialisée, adaptée aux besoins des utilisateurs. De plus, il est souvent nécessaire d'optimiser les assemblages d'algorithmes développés lors de la phase précédente. Cette optimisation peut impliquer l'utilisation de matériel spécialisé (processeurs parallèles, cartes d'acquisition particulières, ...).

Bien que les besoins de ces deux phases peuvent sembler contradictoires, ils ont pourtant un important point de similarité: la nécessité d'un outil capable de s'adapter. Le développement d'un tel outil facilite la création d'un lien direct entre l'application industrielle et l'outil de recherche. Ce lien présente de nombreux avantages en favorisant un transfert technologique entre le laboratoire et l'industrie. Tout d'abord, le développement du produit industriel est accéléré par la réutilisation partielle de la solution développée lors de la phase de recherche. D'autre part, les optimisations développées pour les besoins de l'industrie peuvent être réutilisées au laboratoire, lors du développement d'un nouveau produit.

Depuis quelques années, les applications industrielles du traitement d'images se sont multipliées. Le monde de l'informatique a également beaucoup évolué. Ces facteurs ont mené au développement de SOTISE, un **S**ystème **O**uvert de **T**raitement d'**I**mages **S**ouple et **É**volutif.

Le logiciel SOTISE a originalement été créé pour concevoir un système visuel de contrôle de qualité. Ce logiciel a été développé sous Windows NT et contient déjà plus de 150 routines de traitement d'images.

2 Concepts

Nous croyons que certains principes simples permettent de concilier les besoins de la recherche et ceux de l'industrie. Cette section présente les principaux concepts qui ont servi de moteur au développement de notre système.

2.1 Conception d'un système ouvert

La conception d'un système ouvert est une alternative prometteuse qui aide à développer des applications d'envergure à la fois souples, fiables, faciles à maintenir et capables de s'adapter rapidement aux changements. Cette approche permet de se concentrer uniquement sur l'aspect innovateur d'un logiciel, en réutilisant les outils existants [6], [7].

Dans SOTISE, on utilise les principes de conception d'un système ouvert pour inclure facilement des techniques provenant d'autres domaines (réseaux neuroniques, systèmes experts, ...), ou pour adapter le système à l'industrie (parallélisme [8], contrôle de procédé, ...).

2.2 Utilisation d'un interprète

Le système SOTISE utilise un interprète comme interface au système. L'utilisation d'un interprète, d'une part, permet de créer rapidement de nouveaux assemblages d'algorithmes et, d'autre part, sert de support à l'implantation d'une interface graphique adaptée aux besoins spécifiques des usagers.

Notre interprète joue également d'autres rôles importants: il unifie le système, tel que proposé par Paxson [9], et en brise la complexité. Les exemples d'applications de la section 5 démontrent l'importance de l'interprète dans notre système.

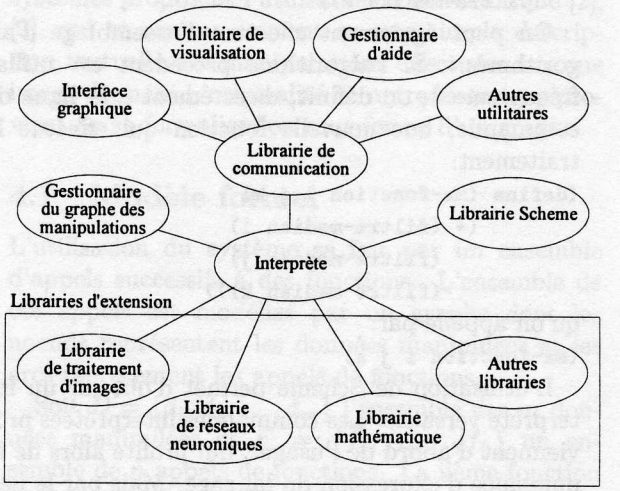


Figure 1: Présentation des principaux modules du système

3 Architecture du système

La figure 1 présente l'architecture du système. Les principaux modules en sont l'interprète, les librairies d'extension, la librairie Scheme, l'interface graphique, l'utilitaire de visualisation, le gestionnaire d'aide et le gestionnaire du graphe des manipulations.

Il n'y a pas de hiérarchie fixée à priori dans SOTISE. Chacun des modules est lié de façon bidirec-

tionnelle à l'interprète.[3] Ces liens sont utilisés pour permettre un échange souple entre tous les modules du système. Les paragraphes suivants décrivent brièvement les principaux modules.

3.1 L'interprète

L'interprète est l'outil privilégié de SOTISE. Cet interprète est une implantation minimale du langage Scheme [10], un langage fonctionnel qui est un dialecte simplifié et rationalisé de LISP. La programmation fonctionnelle propose un paradigme où une fonction est appliquée sur des données, pour obtenir un résultat. Par exemple, la commande

```
(fft (filtre-median i))
```

calcule d'abord le filtre médian de l'image i, évalue ensuite la transformée de Fourier de l'image filtrée et retourne le spectre de l'image comme résultat. Cette approche permet de modéliser des fonctions dont le flot des données est plus qu'un simple pipeline, par exemple:

```
(+ (filtre-median i)
   (filtre-median j)
   (filtre-median k))
```

On peut également effectuer l'assemblage d'algorithmes. Si l'algorithme précédent est utilisé fréquemment, on définit, directement à la ligne de commande, une nouvelle fonction qui effectue le traitement:

```
(define (ma-fonction i j k)
        (+ (filtre-median i)
           (filtre-median j)
           (filtre-median k)))
```

qu'on appelle par:

```
(ma-fonction i j k)
```

L'utilisation de Scheme permet d'obtenir un interprète versatile. Les commandes interprétées proviennent d'abord de l'usager, qui profite alors de la puissance d'expression du langage. Mais par le lien bidirectionnel particulier à SOTISE, cette puissance d'expression s'étend également à tous les modules du système.

3.2 Librairies d'extension

Les librairies d'extension contiennent les structures de données et les algorithmes spécifiques qui étendent les commandes de base de l'interprète. Ces librairies sont écrites en C++ afin de faciliter la réutilisation du code et des structures de données [11]. Ce choix n'a toutefois pas d'influence réelle sur l'architecture globale du système.

Les principales librairies d'extension actuellement utilisées sont:

- Une librairie mathématique qui permet de manipuler les objets mathématiques utiles au traitement d'images: vecteurs, matrices, nombres complexes;
- Une librairie de traitement d'images qui contient les structures de données et les algorithmes. Le tableau 1 détaille les fonctionnalités de cette librairie;
- Une librairie de réseaux neuroniques [12] qui implante les algorithmes spécifiques à un de nos projets;
- Une librairie de communication qui permet l'échange d'information avec les modules externes au système.

L'intégration de nouvelles librairies au système permet d'optimiser certains algorithmes, d'implanter des techniques spécialisées ou encore d'intégrer d'autres logiciels de traitement d'images. Lors de cette intégration des liens seront effectués de façon dynamique entre la nouvelle librairie, l'interprète, le système de documentation, le graphe des manipulations et l'interface graphique. Un script, associé à la librairie, décrit son intégration dans le système.

3.3 Librairie Scheme

Une librairie de fonctions Scheme prédéfinies propose des assemblages standards d'algorithmes et personnalise l'interface en fonction d'un usager ou d'un projet. Les fonctions usagers fréquemment utilisées peuvent se retrouver dans cette librairie. La flexibilité de Scheme permet de créer des fonctions qui n'auraient pas pu être écrites facilement en C++, et plus particulièrement les fonctions qui modifient le comportement de l'interprète [10].

3.4 Interfaces graphiques

L'utilisation d'un même logiciel de traitement d'images à la fois lors de la phase de recherche et développement et lors de l'implantation industrielle conduit à l'utilisation de plusieurs interfaces graphiques radicalement différentes:

- Dans la phase de recherche et développement, on pourrait, par exemple, vouloir créer une interface graphique qui présente les traitements

[3] L'interprète effectue un appel de procédure aux librairies, ce qui correspond à la liaison unidirectionnelle que l'on retrouve généralement dans d'autres systèmes. Les modules du système peuvent également composer une ligne de commande et demander à l'interprète de l'exécuter, ce qui permet alors de créer une liaison bidirectionnelle.

Tableau 1: Algorithmes de traitement d'images du sytème SOTISE

Opérations arithmétiques	Normalisation, facteur d'échelle, etc.
	Addition, soustraction, multiplication, etc.
	Logique (et, ou, ou exclusif), décalage, etc.
Filtres	Convolution bidimensionnelle, médian, LMS, etc.
	Détection de contour, Roberts, Sobel, Robinson, Nevatia-Babu, etc.
Morphologie mathématique	Dilatation, érosion, fermeture, ouverture, filtre morphologique, etc.
Transformées	FFT, DFT, DCT, Walsh, Hartley, etc.
Fenêtres de pondération	Hamming, Hanning, rectangulaire, triangulaire, Blackman
Histogramme	Manipulation, égalisation, etc.
Opérations géométriques	Rotation, transposition, extraction de sous-images, etc.
Segmentation d'images	Seuil, détection de discontinuité (point, ligne, contour), etc.
Analyse d'images	Réseaux neuroniques [12], k-moyennes, corrélation, etc.
Visualisation	1D, 2D et 3D, Visualisation des composantes du spectre de Fourier, etc.
Opérations matricielles	Décomposition en valeurs singulières, valeurs propres et vecteurs propres, déterminant, inverse matricielle, etc.

sous la forme de schémas blocs ou de flux de données, similaire à celles proposées par les langages visuels comme KBVision, Khoros [4] ou WiT;

- Dans la phase d'implantation industrielle, l'interface doit répondre aux besoins spécifiques du client. Ce dernier pourrait demander, par exemple, une interface qui présente le contrôle de processus industriels effectué par le système de traitement d'images, tout en cachant les détails des algorithmes de traitement.

SOTISE possède la capacité de supporter plusieurs interfaces graphiques différentes, et spécifiques à des projets. Ce support est effectué par le biais de l'interprète. Les différentes interfaces peuvent être écrites avec des outils existants tels que ToolBook ou Tcl/tk. Elles répondent aux requêtes de l'usager par l'envoi de commandes à l'interprète et réagissent de la façon appropriée aux réponses de ce dernier.

3.5 Graphe des manipulations

Le graphe des manipulations est un concept original et important dans le système SOTISE. Il est décrit à la section suivante.

4 Graphe des manipulations

Le traitement d'images implique souvent un grand nombre d'algorithmes et de manipulations successives d'images. Un aspect important d'un logiciel de traitement d'images est sa capacité d'aider à gérer l'ensemble des traitements effectués. Plusieurs systèmes proposent l'utilisation d'un historique [2], [3], qui se résume souvent en une simple description textuelle des manipulations effectuées sur une image donnée. L'originalité de notre historique provient de sa formalisation sous forme d'un graphe.

4.1 Modèle formel

L'utilisation du système se fait par un ensemble d'appels successifs à des fonctions. L'ensemble de ces appels est modélisé par un graphe dont les noeuds représentent les données manipulées et les arcs représentent les appels de fonctions.

Soit $D = \{d_1, d_2, \ldots, d_m\}$ l'ensemble des m données manipulées et $F = \{f_1, f_2, \ldots, f_n\}$ un ensemble de n appels de fonctions. La iième fonction $f_i \in F$ est définie par:

$$f_i(d_{S_1^i}, d_{S_2^i}, \ldots, d_{S_{P^i}^i}) = d_{R^i}$$

où S_j^i est l'indice du jième paramètre, R^i l'indice du résultat et P^i le nombre de paramètres de la fonction f_i, tel que $d_{R^i} \in D$ et $d_{S_j^i} \in D$. On modélise cet ensemble d'appels de fonctions par le graphe ordonné $G = [V, E]$.

L'ensemble des noeuds V est défini par $D \cup \{v_1, v_2, \ldots, v_n\}$ où v_i est un noeud supplémentaire introduit pour chaque fonction $f_i \in F$, afin de simplifier le modèle. Dans le cas des fonctions à paramètres multiples, ce noeud permet d'associer le coût d'une fonction à un arc unique plutôt qu'à un ensemble d'arcs. Les noeuds v_3 et v_5 de l'exemple

présenté plus loin dans le texte (voir figure 2), permettent d'associer le coût des fonctions f_3 et f_5 aux arcs (v_3, d_5) et (v_5, d_6).

L'ensemble des arcs E lie les paramètres d'une fonction f_i à son résultat, par l'intermédiaire du noeud v_i. Il est défini par:

$E = \{(d_{S_j^i}, v_i) : 1 \leq i \leq n, 1 \leq j \leq P^i\} \cup$
$\{(v_i, d_{R^i}) : 1 \leq i \leq n\}$

À chaque appel f_i par l'interprète, les noeuds et les arcs correspondants seront ajoutés dynamiquement dans le graphe, d'une façon transparente à l'utilisateur. Certaines informations peuvent également être associées aux noeuds ou aux arcs: temps d'exécution d'une fonction, mémoire nécessaire pour conserver une donnée, appréciation personnelle de l'utilisateur quant au résultat d'un traitement, etc.

Soit la définition d'une fonction Scheme qui effectue une convolution par l'intermédiaire d'une transformée de Fourier:

```
(define (filtrage-fft image coef-filtre)
  (fft-inverse (mult-terme (fft image)
                           (fft coef-filtre))))
```

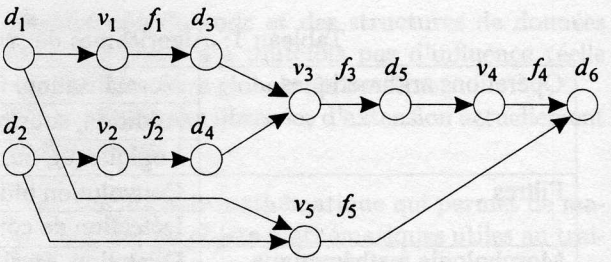
où `fft` calcule la transformée de Fourier rapide, `fft-inverse` la transformée inverse et `mult-terme` une multiplication terme à terme. La figure 2 présente le graphe correspondant à l'appel `(filtrage-fft i j)`. La définition de la fonction `filtrage-fft` est un niveau d'abstraction proposé par l'usager. Le traitement est représenté à deux niveaux d'abstraction différents: soit par la fonction `(filtrage-fft i j)`, ou encore par la fonction équivalente `(fft-inverse (mult-terme (fft i) (fft j)))`. L'information associée à ces niveaux d'abstraction se retrouve dans le graphe par des chemins multiples de d_1 et d_2 vers d_6 (voir figure 2). Un premier chemin décrit l'appel de la fonction `(filtrage-fft i j)`, alors qu'un deuxième chemin décrit un appel de fonction équivalent, mais plus détaillé, `(fft-inverse (mult-terme (fft i) (fft j)))`.

4.2 Utilisation du graphe

L'application d'algorithmes de recherche opérationnelle sur le graphe des manipulations [13] permet de résoudre plusieurs problèmes. Quatre de ces problèmes seront exposés dans cette section.

4.2.1 Historique des traitements

L'historique d'une donnée d_k est représenté par l'ensemble des traitements effectués pour obtenir d_k. On définit cet historique par le sous-graphe $G' = [V', E']$, où V' est l'ensemble de tous les



f_1: `fft`$(d_1) = d_3$
f_2: `fft`$(d_2) = d_4$
f_4: `fft-inverse`$(d_5) = d_6$
f_5: `filtrage-fft`$(d_1, d_2) = d_6$
f_3: `mult-terme`$(d_3, d_4) = d_5$

Figure 2: Graphe des manipulations d'un filtrage via une FFT

noeuds d_i tel qu'il existe un chemin de d_i à d_k dans G, et où l'ensemble des arcs E' est définit par $E' = \{(a, b) \in E : a \in V' \wedge b \in V'\}$.

Deux techniques permettent de visualiser l'historique d'une donnée. La première technique consiste à afficher le sous-graphe G'. La seconde technique consiste à observer les relations d'un noeud vers ses voisins, et à naviguer d'un noeud à l'autre par des liens hyper-texte. Dans ce cas, il n'est pas nécessaire de calculer entièrement le sous-graphe G'.

Nous avons implanté la seconde technique à l'aide de l'utilitaire d'aide de Windows NT, mais cette technique s'utilise tout aussi bien avec d'autres logiciels de navigation hyper-texte, comme Mosaic ou Netscape.

4.2.2 Réexécution

Soit $C \subseteq D$ l'ensemble des données actives dans le système (conservées en mémoire ou sur le disque) et d_k une donnée inactive du système. Si une recherche dans l'historique de d_k permet d'obtenir une fonction $f(\alpha_1, \alpha_2, \ldots, \alpha_p) = d_k$, où $\alpha_i \in C$ pour $1 \leq i \leq p$, alors l'appel de la fonction $f(\alpha_1, \alpha_2, \ldots, \alpha_p)$ est appelée réexécution de d_k.

La réexécution est appliquée dans diverses situations. Elle est d'abord utilisée pour visualiser une image lors de la présentation de l'historique. Le problème est trivial lorsque l'image est une donnée active du système. Lorsque l'image est une donnée inactive, sa réexécution permet de l'afficher.

La réexécution est également utilisée pour vérifier rapidement la validité d'une technique de traitement d'image. Soit $f(\alpha_1, \alpha_2, \ldots, \alpha_p) = d_k$ une fonction qui correspond à une technique appliquée aux données $\alpha_1, \alpha_2, \ldots, \alpha_p$. Alors l'appel $f(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \beta_i, \alpha i + 1, \ldots, \alpha_p) = d_l$ permet

d'évaluer la technique avec la nouvelle donnée β_i.

Une réexécution peut finalement être optimisée en fonction de critères précis: espace mémoire utilisé, temps de traitement, etc. Cette possibilité est particulièrement intéressante lors de l'implantation industrielle d'un algorithme.

4.2.3 Gestion de l'espace disque

En traitement d'images, un des problèmes souvent rencontré est la quantité énorme d'espace disque utilisé (ex: *remote sensing*). Le graphe peut fournir une solution à ce problème par la gestion d'un disque virtuel. Dans un tel disque virtuel, il est possible de gagner de l'espace en compressant certaines images, en contrôlant la création de sauvegardes, ou encore en détruisant les images qui peuvent être reproduites par la réexécution.

4.2.4 Partage d'information

Dans le cas où plusieurs personnes travaillent en collaboration sur un même projet, il est intéressant de partager un graphe commun, représentant l'ensemble des traitements effectués par l'équipe. Ce graphe donne alors accès à l'historique de toutes les images manipulées par l'équipe et à toutes les fonctions de réexécution de ces images. L'utilisation d'un graphe unique, accédé par le biais d'appels de procédure à distance (RPC), assure la cohérence de l'information représentée.

4.3 Coût du graphe

Le coût de la création et de l'utilisation du graphe des manipulations peut d'abord sembler élevé. En fait, l'appel d'une fonction interprétée n'implique qu'un appel de fonction au gestionnaire du graphe des manipulations et l'ajout potentiel de quelques noeuds (1 ou 2), et quelques arcs (3 ou 4). Ces ajouts sont effectués dynamiquement, de façon efficace, sans recherche dans le graphe. Comme le graphe est mis à jour lors d'un appel interprété, le coût de création du graphe est négligeable par rapport au temps d'application des algorithmes de traitement d'images. Un noeud et un arc, sans information particulière, utilisent tous deux 16 bytes. Le temps de calcul d'une fonction, les commentaires de l'utilisateur sur le résultat d'un traitement particulier ou d'autres informations pourront augmenter l'espace mémoire utilisé. Les algorithmes appliqués sur le graphe pour obtenir des informations précises sont de coût plus élevé. La structure particulière du graphe se prête cependant à plusieurs hypothèses simplificatrices qui permettent d'utiliser des algorithmes efficaces. Soit n le nombre de noeuds et m le nombre d'arcs du graphe des manipulations. Comme ce graphe est acyclique, il est possible de trouver le plus court chemin entre deux noeuds dans $O(m)$, et d'un noeud vers tous les autres ou de tous les noeuds vers un noeud, dans $O(nm)$ [13]. Ces deux algorithmes permettront de résoudre efficacement les problèmes exposés.

5 Exemples d'application

Nous avons présenté les concepts qui ont mené à la conception du système, les différents modules de SOTISE et, finalement, le graphe des manipulations et quelques-unes de ses applications. L'article se termine par des exemples qui illustrent l'application de ces concepts confrontés à des problèmes concrets.

5.1 Utilitaire de visualisation

Deux solutions sont généralement proposées pour implanter un utilitaire de visualisation [3], [4]. Avec la première solution, l'utilitaire gère tous les détails de l'affichage. Cette approche monolithique implique une dépendance entre les algorithmes et le programme d'affichage: par exemple, pour afficher correctement une transformée de Fourier, le programme d'affichage devrait avoir la possibilité de manipuler l'image pour présenter la transformée optique ou ordinaire. Avec la deuxième solution, l'utilitaire est entièrement indépendant de l'algorithme mais il est alors plus difficile de contrôler de façon interactive la visualisation.

Nous proposons une troisième alternative. Dans notre système, le programme d'affichage est indépendant de l'algorithme. La communication bidirectionnelle entre le programme de visualisation et l'interprète permet toutefois d'implanter un contrôle interactif de la visualisation (voir figure 3). Lorsque le programme de visualisation est lancé il n'a aucune connaissance spécifique sur l'algorithme utilisé. C'est l'interprète qui lui communique une séquence d'options d'affichage: gestion de la table des couleurs, pile d'images [5], accès à l'historique, utilisation d'un curseur, etc. Le programme de visualisation propose ces options à l'usager puis répond aux requêtes de l'usager en communiquant de nouveau avec l'interprète pour qu'il exécute la fonction associée.

Cette approche permet de briser la complexité du système dans le sens où l'utilitaire de visualisation ne possède qu'une coquille rudimentaire d'affichage

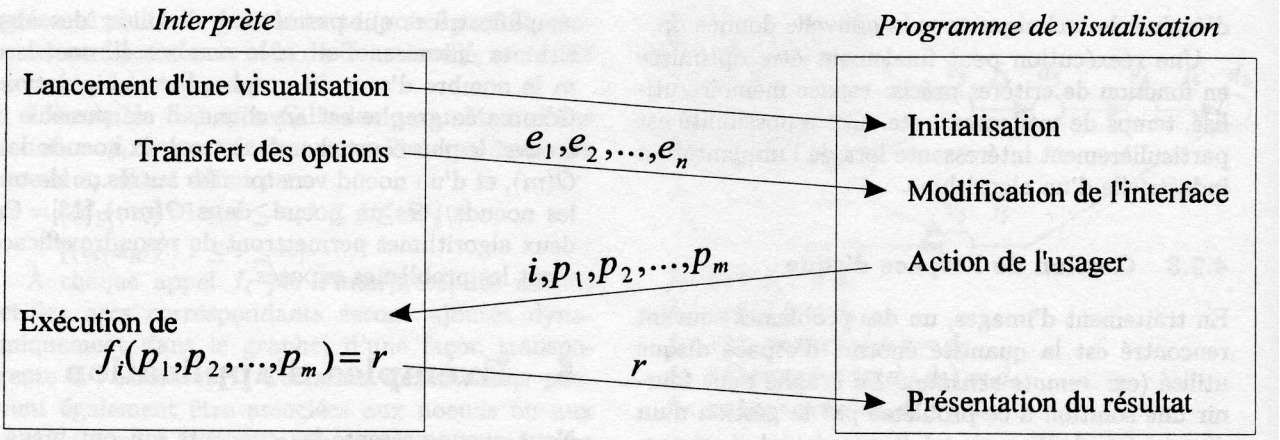


Figure 3: Évènements associés à une visualisation.
Cette figure décrit la séquence des évènements qui se produisent lors d'une visualisation. Une option d'affichage est une paire (e_i, f_i) où e_i est une description de l'option et f_i décrit la fonction qui sera exécutée si l'option est sélectionnée. e_i décrit i) les modifications à apporter à l'interface du programme de visualisation (ajouter une option dans le menu, permettre l'utilisation du curseur, ...), ii) les paramètres $p_1, p_2, \ldots, p_m$ à transférer à l'interprète si cette option est sélectionnée (position du curseur, entrée de l'usager, ...), et iii) l'action associée au résultat r retourné (afficher l'information numérique, afficher une nouvelle image, ...).

et un moyen de communication avec l'interprète. Pour l'usager, le tout est parfaitement transparent.

5.2 Visualisation d'un espace de paramètres

L'exemple présenté dans cette section est très spécifique, mais introduit des concepts qui s'appliquent à plusieurs autres problèmes de traitement d'images.

La discrimination de deux classes d'images (ou de sous-images) est un problème classique en reconnaissance de forme. Une des techniques applicables à ce problème consiste à trouver des paramètres significatifs qui permettront l'utilisation de fonctions discriminantes. Il est alors commode de visualiser la distribution des classes en fonction des valeurs de ces paramètres (espace de paramètres). L'interprétation visuelle des nuages de points du graphique nous donne une idée de la séparabilité des classes et du pouvoir discriminant des paramètres utilisés. Dans le cas d'un espace de n paramètres, il est toujours possible d'utiliser un algorithme de projection optimale [14] pour se ramener à un espace à 2 ou 3 dimensions.

De façon plus formelle, soit un ensemble de m sous-fenêtres $S = \{s_1, s_2, \ldots, s_m\}$ et un ensemble de tuples $T = \{p_1, p_2, \ldots, p_m\}$, où $p_i = (x_1^i, x_2^i, \ldots, x_n^i)$ est un point dans un espace à n dimensions et x_j^i la valeur du jième paramètre de s_i. L'évaluation des paramètres est définie comme l'application des fonctions de paramètres $f_j(s_i) = x_j^i$ pour $1 \leq i \leq m$ et $1 \leq j \leq n$.

L'analyse de la pertinence des paramètres est effectuée en trois étapes: la définition de l'ensemble S, le calcul de l'ensemble des paramètres T et la présentation graphique de cet espace. Dans cet exemple, nous n'allons nous intéresser qu'à l'étape du calcul de l'ensemble des paramètres.

Dans les logiciels d'analyse d'images, deux approches sont typiquement utilisées:

(1) Solution dédiée d'analyse: les fonctions de paramètres f_i sont prédéfinies;

(2) Interprétation des fonctions de paramètres: l'usager peut définir les fonctions f_i.

L'implantation de la première approche est triviale, efficace et effectuée entièrement dans une librairie de traitement d'images. La seconde approche utilise les fonctions d'ordre supérieur introduites par les langages fonctionnels [10]. Une fonction d'ordre supérieur manipule elle-même d'autres fonctions. L'implantation de cette approche se fait directement à partir de cette définition: les arguments à la fonction de calcul de paramètre sont les fonctions f_i utilisées et l'ensemble S. Cette approche est flexible mais coûteuse, car il faut interpréter chacune des fonctions.

L'architecture particulière de notre système permet d'envisager une implantation efficace de l'approche (2). La distinction entre les approches (1) et

(2) provient de la séparation qui existe généralement entre la librairie et l'interprète dans un système. Initialement, le système est dans un mode interprété qui permet beaucoup de flexibilité mais présente un plus haut coût d'utilisation. Lorsque l'on passe à la librairie, il y a une perte de flexibilité, mais un gain d'efficacité. Le passage est unidirectionnel: de l'interprète à la librairie.

Notre architecture permet d'effectuer un passage bidirectionnel de l'interprète à la librairie. C'est cette possibilité qui est exploitée pour résoudre le problème. La fonction appelée est une fonction d'ordre supérieur. Le passage bidirectionnel de l'interface à la librairie permet toutefois d'intégrer une telle fonction à la librairie, ce qui permet d'éviter les problèmes de performance de l'approche (2). Lorsqu'une fonction de calcul de paramètre provient de la librairie, on effectue directement l'appel. Si toutes les fonctions de calcul proviennent de la librairie, notre implantation est aussi efficace que l'approche (1). Lorsqu'une fonction de calcul de paramètre est définie par l'usager, la librairie effectue un appel à l'interprète afin qu'il l'évalue. Dans ce cas, il y a une perte au niveau de l'efficacité, mais un gain de flexibilité.

SOTISE permet de manipuler directement les ensembles S et T. La fonction de conversion:

```
(tset->tuples S f1 f2 ... fn)
```

applique les fonctions f_i à chacune des m sous-fenêtres de S pour obtenir un ensemble T. Nous allons présenter trois exemples d'utilisation de cette fonction.

i) Utilisation de fonctions de la librairie comme paramètre:

Cet exemple utilise la moyenne et la variance pour générer un espace de paramètres à deux dimensions. Soit S un ensemble de sous-fenêtres et $p_i = (x_1^i, x_2^i)$ un point de l'espace de paramètres correspondant à la sous-fenêtre s_i de S, alors `moyenne`$(s_i) = x_1^i$ et `variance`$(s_i) = x_2^i$:

```
(tset->tuples S moyenne variance)
```

L'interprète appelle la librairie pour exécuter la fonction et toute l'exécution se fait au niveau de la librairie puisque moyenne et variance sont des fonctions internes.

ii) Utilisation de fonctions usager comme paramètre:

Dans le second exemple, l'usager définit une fonction spécifique pour calculer la composante continue du spectre de Fourier:

```
(define (cc s) (harmonique (fft s) 0 0))
```

et cette fonction est ensuite utilisée pour générer un espace de paramètres.

Soit $p_i = (x_1^i, x_2^i, x_3^i)$ un point de l'espace de paramètre correspondant à la sous-fenêtre s_i de S, alors `moyenne`$(s_i) = x_1^i$, `variance`$(s_i) = x_2^i$ et `cc`$(s_i) = x_3^i$:

```
(tset->tuples S moyenne variance cc)
```

La moyenne et la variance seront évaluées directement dans la librairie, tandis que la fonction `cc` est évaluée par un appel à l'interprète. L'interprète appelle récursivement la librairie pour évaluer les fonctions `harmonique` et `fft`.

iii) Création dynamique de fonctions:

Dans le dernier exemple nous utilisons plusieurs descripteurs de Fourier comme paramètres. Une première solution au problème, similaire à celle de l'exemple ii), serait de définir une fonction pour chaque descripteur de Fourier. Une solution plus élégante utilise les fonctions d'ordre supérieur pour résoudre le problème.

Nous commençons par implanter le concept des descripteurs de Fourier en généralisant la fonction `cc` de l'exemple précédant:

```
(define (df s x y) (harmonique (fft s) x y))
```

Cet implantation de la fonction `df` est très naïve car la transformée de Fourier serait calculée à chaque appel. Il est facile de l'implanter de telle façon que cette transformée ne soit calculée qu'une seule fois pour tous les descripteurs de Fourier d'une même image, mais c'est hors du sujet de cet exemple. La fonction `df` ne peut toutefois pas être utilisée directement par la fonction `tset->tuples`, puisqu'elle prend trois arguments.

En Scheme, une fonction retourne une autre fonction par l'utilisation de lambda. Cette possibilité est utilisée pour créer dynamiquement une fonction de s:

```
(define (cdf x y) (lambda (s) (df s x y)))
```

L'appel `(cdf 0 0)` retourne une fonction qui évalue le descripteur de Fourier à l'harmonique `(0,0)` pour une image. L'appel de `((cdf 0 0) s)` retourne donc la valeur du descripteur de Fourier à l'harmonique `(0,0)` de l'image s. Comme la fonction retournée par un appel à `cdf` ne prend plus qu'une sous-fenêtre comme paramètre, elle peut être utilisée dans un appel à `tset->tuples`.

Nous utiliserons la fonction `cdf` pour générer un espace de paramètres à 9 dimensions, où chaque dimension correspond à un descripteur de Fourier spécifique. Soit $p_i = (x_1^i, x_2^i, \ldots, x_9^i)$ un point de l'espace de paramètre correspondant à la sous-fenêtre s_i de S, alors x_j^i correspond à la valeur d'un descripteur de Fourier:

```
(tset->tuples S (cdf 0 0) (cdf 0 1) (cdf 0 2)
                (cdf 1 0) (cdf 1 1) (cdf 1 2)
                (cdf 2 0) (cdf 2 1) (cdf 2 2))
```

Les fonctions résultant des appels successifs à `cdf` seront alors évaluées par des appels à l'interprète.

Ce dernier exemple présente une implantation d'un algorithme de classification basé sur l'utilisation de descripteurs de Fourier. L'implantation d'un tel algorithme, en trois lignes, à partir d'un système qui ne contient pas de notions de descripteurs de Fourier est un exemple de la flexibilité de SOTISE.

6 Conclusion

Nous avons présenté les grandes lignes de l'application de quelques concepts à un logiciel de traitement d'images. Le graphe des manipulations est un concept original et prometteur qui offre la possibilité de raisonner sur l'ensemble des traitements effectués et de résoudre ainsi une vaste gamme de problèmes. Notre expérience nous montre que l'approche utilisée offre la flexibilité nécessaire pour s'adapter aux besoins de la recherche et de l'industrie. L'implantation d'un prototype de système visuel de contrôle de qualité nous a permis de constater cette flexibilité. Notre philosophie, appliquée à la conception d'outils de traitement d'images, est tout à fait originale dans le domaine et trace sans doute une voie à suivre.

Références

[1] M.J.B. Duff, and S. Levialdi, "Languages and Architectures for Image Processing", *Academic Press Inc.*, London, 1981.

[2] C. Roberge, "Reconnaissance et localisation spatiale des objets dans une scène tridimensionnelle", *Thèse de doctorat (Ph.D.)*, Université de Sherbrooke, juin, 1990.

[3] M.S. Landy, "The HIPS-2 Software for Image Processing: Goals and directions", *SPIE* Vol. 1964, Applications of Artificial Intelligence, pp. 382–391, 1993.

[4] K. Konstantinides and J.R. Rasure, "The Khoros Software Development Environment for Image and Signal Processing", *IEEE Trans. Image Processing*, Vol. 3, no. 3, pp. 243–252, May 1994.

[5] D. Heeger, E. Simoncelli and E.J. Chichilnisky, "OBVIUS: Object-Based Vision and Image Understanding System", Vision Science Group, Media Laboratory, MIT, Mass.,1993.

[6] P. Calder, and M. Linton, "The Object-Oriented Implementation of a Document Editor", *Proc. OOPSLA '92*, Vancouver, B.C., pp. 154–165, Oct. 18–22, 1992.

[7] I. Jacobson, "Object Oriented Development in an Industrial Environnement", *Proc. OOPSLA '87*, Orlando, Florida, pp. 183–191, Oct. 4–8, 1987.

[8] L.H. Jamieson, E.J. Delp, C.-C. Wang, J. Li and F.J. Weil, "A Software Environment for Parallel Computer Vision", *IEEE Computer*, Vol. 25, No 2, pp. 73–77, Feb. 1992.

[9] V. Paxson and C. Saltmarsh, "Glish: A User-Level Software Bus for Loosely-Coupled Distributed Systems", *Proc. 1993 Winter USENIX Conference*, San Diego, CA, Jan., 1993.

[10] J.A. Rees and W. Clinger (editors), "Revised^4 Report on the Algorithmic Language Scheme", MIT AI Memo 848b, Cambridge, Mass., Nov., 1991.

[11] B. Stroustrup, "The Design and Evolution of C++", Addison Wesley, Reading, Mass. 1994.

[12] A.G. Supynuk and W.W. Armstrong, "Adaptive Logic Networks and Robot Control", *Proc. Vision Interface Conference '92*, Vancouver, B.C., pp. 181–186, May 11–15, 1992.

[13] R.E. Tarjan, "Data Structures and Network Algorithms", *SIAM*, Philadelphia, 1983.

[14] M.D. Vanstrum and Strarks, "An Algorithm for Optimal Maps", *Proc. IEEE Int. Symp. Inform. Theory*, Feb., 1981.