utbm
université de technologie
Belfort-Montbéliard

CARRE Julien

PIERNOT Jérôme

Rapport IN52

Sujet : Résolution d'un puzzle

Responsable : M. Ruicheck Y.

Automne 2007

SOMMAIRE

INTRODUCTION..3

I. Description et approche du sujet...4

1. Description
2. Outils utilisés

II. Architecture………...5

III. Techniques utilisées...7

1. Découpage de l'image
2. Comparaison par couleur………...8
3. Comparaison par formes………...10

IV. Exemples………...12

CONCLUSION………...16

INTRODUCTION

Ce projet a été réalisé dans le cadre de l'UV IN52, dans le but de pouvoir utiliser toutes les notions liées à la détection de formes vues au cours de ce semestre.

Le but de ce projet est de réaliser un solveur de puzzle entièrement automatisé par l'ordinateur en utilisant diverses notions de traitement d'images.

Nous allons tout d'abord décrire l'intitulé de notre sujet et notre approche vis-à-vis de celui-ci. Ensuite, nous parlerons plus en détails des techniques utilisées afin d'arriver à un résultat. Par la suite nous parlerons de l'architecture que nous avons employée au niveau du code. Enfin, nous terminerons en présentant un exemple.

I. Description et approche du sujet

1. Description

Le sujet que nous avons choisi est la résolution d'un puzzle. Il nous a été demandé de mettre en application certaines notions utilisant la détection de contours de manière à ce que le logiciel puisse résoudre automatiquement un puzzle.

Nous nous sommes alors poser plusieurs questions permettant de cerner au mieux le projet.
La première a été de savoir si le logiciel doit fournir la possibilité de pouvoir réaliser la résolution d'un puzzle à partir d'une image ou de plusieurs en laissant le choix à l'utilisateur de l'image. Nous avons décidé que l'utilisateur pouvait choisir l'image dont il souhaitait la résolution du puzzle, ceci afin de rendre l'utilisation du logiciel plus agréable en proposant à chaque fois un nouveau puzzle.

Le nombre de pièces du puzzle a été la deuxième interrogation, est-il possible de définir un nombre de pièces ou de mettre un nombre par défaut. Dans la continuité de ce qui a été vu auparavant, nous pensons que le fait de permettre à l'utilisateur de choisir le nombre de pièces était plus conviviale.

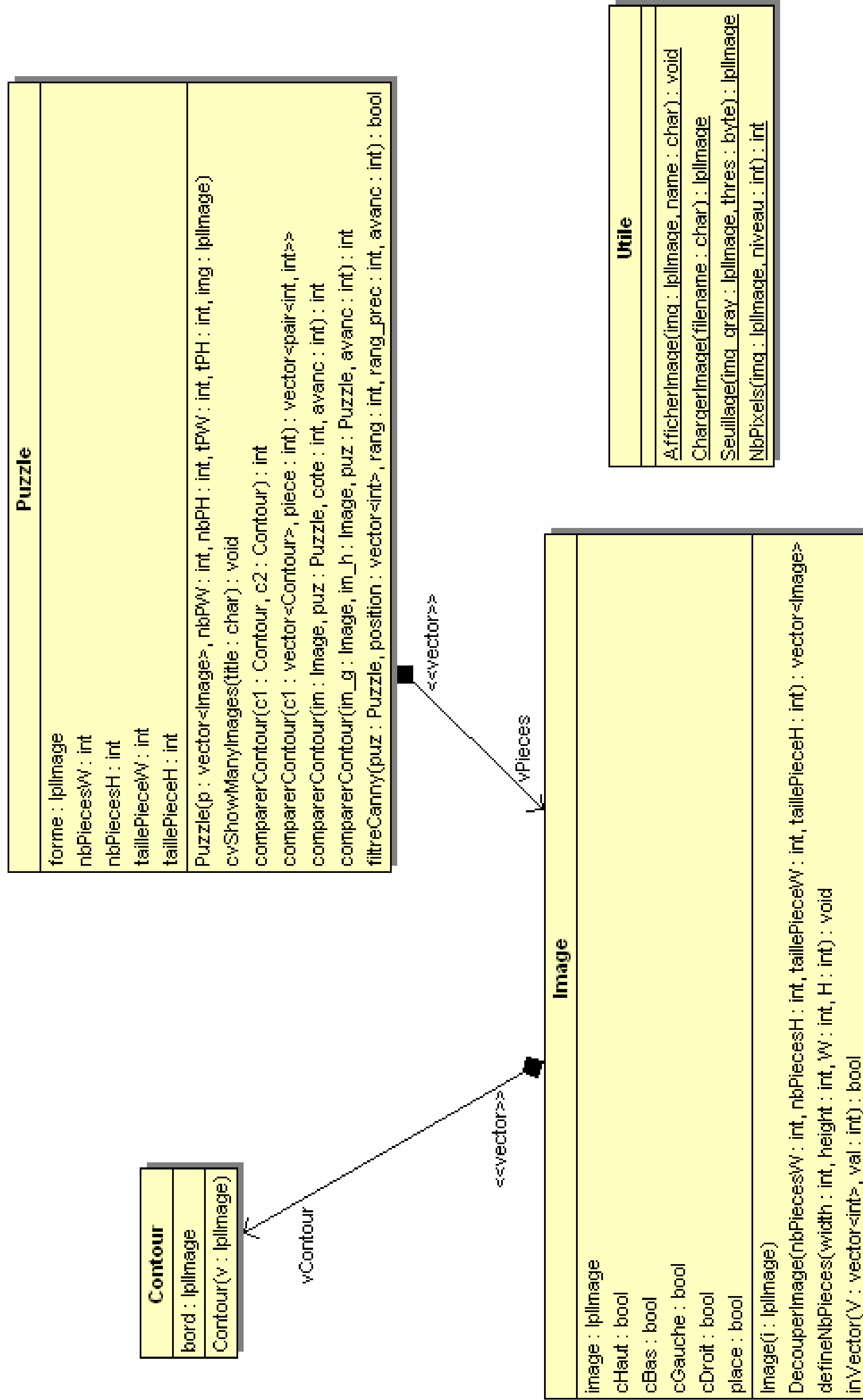
Nous nous sommes ensuite interrogés sur l'image en elle-même et surtout sur le fait de limiter le logiciel à des images en niveau de gris ou si l'utilisation de la couleur pouvait être possible.
Etant donné que nous n'avons utilisé que des images en niveaux de gris au cours des TP, nous nous sommes dit qu'il serait plus bénéfiques pour nous de pouvoir travailler à présent sur tout types d'images, et cela permet également à l'utilisateur de pouvoir utiliser n'importe quel type d'image.

Toujours dans une problématique vis-à-vis de l'utilisateur, nous nous sommes demandés quelles étaient les informations à afficher.
Nous avons décidé d'afficher le minimum d'informations, par conséquent aucune interface graphique ne sera présente, on affiche l'image choisie par l'utilisateur, puis celui-ci renseignera le nombre de pièces dans la console. Une fenêtre affichera alors le puzzle généré avec les pièces dans le désordre et une fois le traitement effectué, une nouvelle fenêtre s'affichera avec le puzzle résolu.

2. Outils utilisés

Nous avons décidé de développer notre application à l'aide du logiciel Microsoft Visual C++ 2005 Express Edition étant donné que c'est un logiciel gratuit et celui utilisé en TP.
Nous avons également utilisé OpenCV qui est une bibliothèque gratuite de traitement d'images.
OpenCV signifie Intel R Open Source Computer Vision Library.
Cette bibliothèque a été créée par Intel Corporation. Il s'agit d'une collection de près de 300 fonctions et classes en langage C et C++.

II. Architecture



L’image d’origine correspond à une Image ainsi que chaque pièce du puzzle. Nous créons chacune des pièces à l’aide de la méthode DecouperImage.

Les contours de chaque pièces sont crées dans le constructeur de Image, un vector<Contour> est dont présent pour chaque pièce.

Un puzzle est ensuite crée en passant un vector<Image> correspondant à chaque pièce. On passe également en paramètre le nombre de pièces ainsi que la taille en hauteur et largeur et enfin l’image correspondant aux contours de l’image d’origine.

La méthode CreerForme() de Image permet de créer l’image correspondant aux contours des formes.

La méthode comparerContour de Puzzle permet de comparer les bords des pièces avec les bords correspondant des autres pièces.

La méthode filtreCanny permet de comparer les formes entre deux pièces.

III. Techniques utilisées

1. Decoupage de l'image

La première technique que nous avons utilisée a été le découpage de l'image.
Nous avons souhaité que l'utilisateur puisse choisir le nombre de pièces du puzzle que ce soit en largeur comme en hauteur de manière à avoir le plus large choix de puzzle possible.
Pour cela, il nous fallait tout d'abord rechercher les multiples en hauteur et largeur de l'image. De cette manière nous pouvons fournir un choix de pièces à l'utilisateur.

```
//recherche des multiples
      for ( int i=2; i<width/10; i++ )
      {
            if ( width%i == 0 ) Vwidth.push_back (i);
      }

      for (int i=2; i<height/10; i++ )
      {
            if ( height%i == 0 )
            {
                  Vheight.push_back (i);
            }
      }
```

Concernant le découpage de l'image, une fois le nombre de pièces connu en largeur et hauteur ainsi que leur taille, nous avons choisi de ne découper les bords que sur un pixel, nous procédons de la facon suivante :

```
      for(int i=0; i<nbPiecesH;i++)
      {
            for(int j=0; j<nbPiecesW;j++)
            {
                  IplImage* imgtmp =
                  cvCreateImage(cvSize(taillePieceW,taillePieceH),depth
                  ,channels);
                  cvGetRectSubPix(image,imgtmp,
                  cvPoint2D32f((taillePieceW/2)+
                  j*taillePieceW,(taillePieceH/2)+i*taillePieceH));
                  Image im(imgtmp);
            }
      }
```

La fonction cvGetRectSubPix permet d'extraire une partie de l'image suivant un rectangle donc le centre est donné en paramètre de la fonction.

Pour mélanger le puzzle, on choisit aléatoirement les pièces crées auparavant à l'aide de la fonction rand().

2. Comparaison par couleur

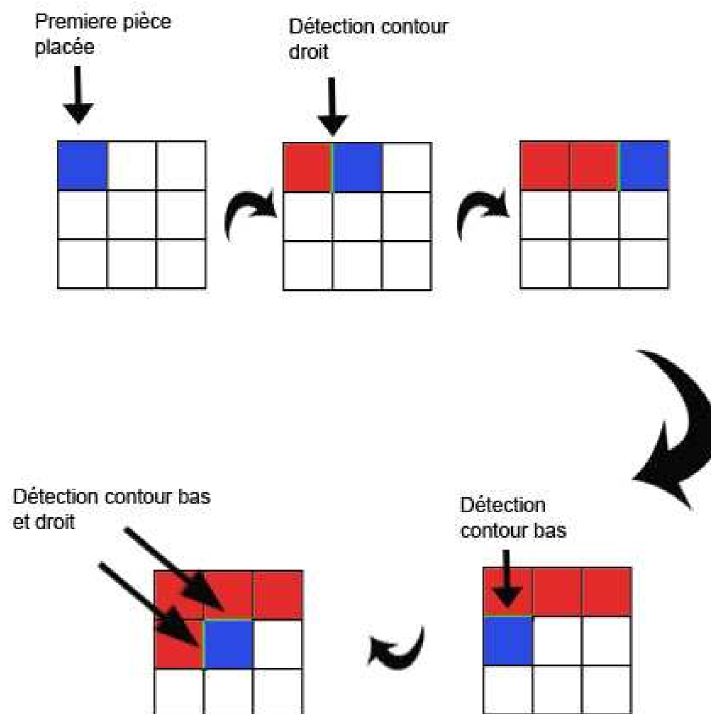
Notre première méthode afin de reformer le puzzle a été de comparer chaque bord d'une pièce avec un autre bord et de comparer l'intensité des pixels.

Il nous a fallu tout d'abord découper les bords. Pour cela, nous avons utilisé la même fonction qu'auparavant cvGetRectSubPix en l'adaptant suivant le bord à couper (Haut, Bas, Gauche, Droit).
Pour chaque pièce traitée, nous enregistrons également s'il s'agit d'une pièce appartenant au bord du puzzle. Ensuite nous traitons la résolution du puzzle de la manière suivante :

Nous cherchons la pièce se situant en haut à gauche (première pièce du puzzle). A partir de celle-ci, nous recherchons la pièce se situant à sa droite à l'aide d'une fonction comparerContour que nous préciserons par la suite. Nous procédons de la même manière sur la pièce trouvée et ainsi de suite afin de compléter la première ligne.

Pour ce qui est des lignes suivantes, nous recherchons la pièce se situant en bas de la première pièce de la ligne precedente. Puis nous recherchons la pièce à droite en comparant cette fois-ci sur les contours droit de la pièce trouvée et haut de la pièce de la ligne précedente.

On effectue alors ce traitement jusqu'à ce que toutes les pièces soient traitées, en spécifiant à chaque fois que la pièce a bien été placée.

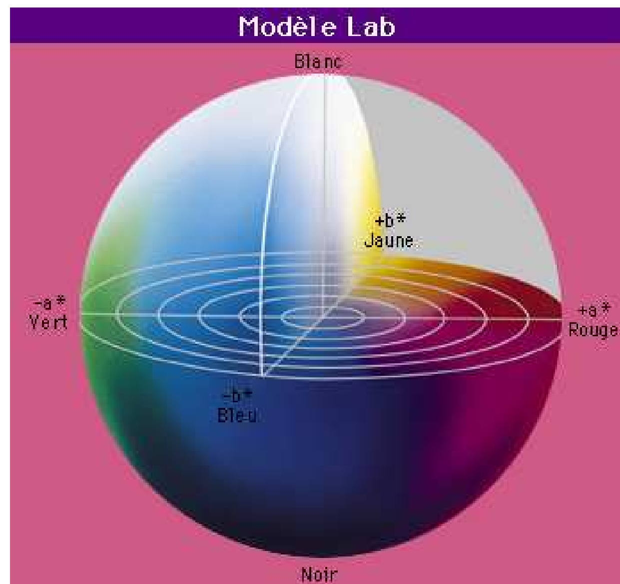


Nous avons décidé de comparer les contours de chaque pièce en utilisant les couleurs. Notre image d'origine étant en codage couleur RGB, nous avons décidé d'utiliser le codage CIE Lab pour effectuer nos traitements. **CIE Lab** est un modèle de représentation des couleurs développé par la commission Internationale de l'Éclairage (CIE), en 1976.

- La combinaison **L*** est la clarté, qui va de 0 (noir) à 100 (blanc).
- La composante **a*** représente la gamme de l'axe rouge (valeur positive) -> vert (négative) en passant par le blanc (0) si la clarté vaut 100.
- La composante **b*** représente la gamme de l'axe jaune (valeur positive) -> bleu (négative) en passant par le blanc (0) si la clarté vaut 100.

Nous avons décidé d'utiliser ce codage car il essaye de prendre en compte la réponse logarithmique de l'oeil. Le but de cette méthode est de calculer la distance entre deux pixels en termes de couleur or deux couleurs qui sont proches dans l'espace de couleur (au sens de la distance euclidienne le plus souvent), peuvent paraître assez différentes pour l'oeil, ce qui est le cas pour l'espace des couleurs RGB.

Par contre, dans l'espace LAB qui est uniforme, deux couleurs proches en distance le sont aussi pour l'oeil.



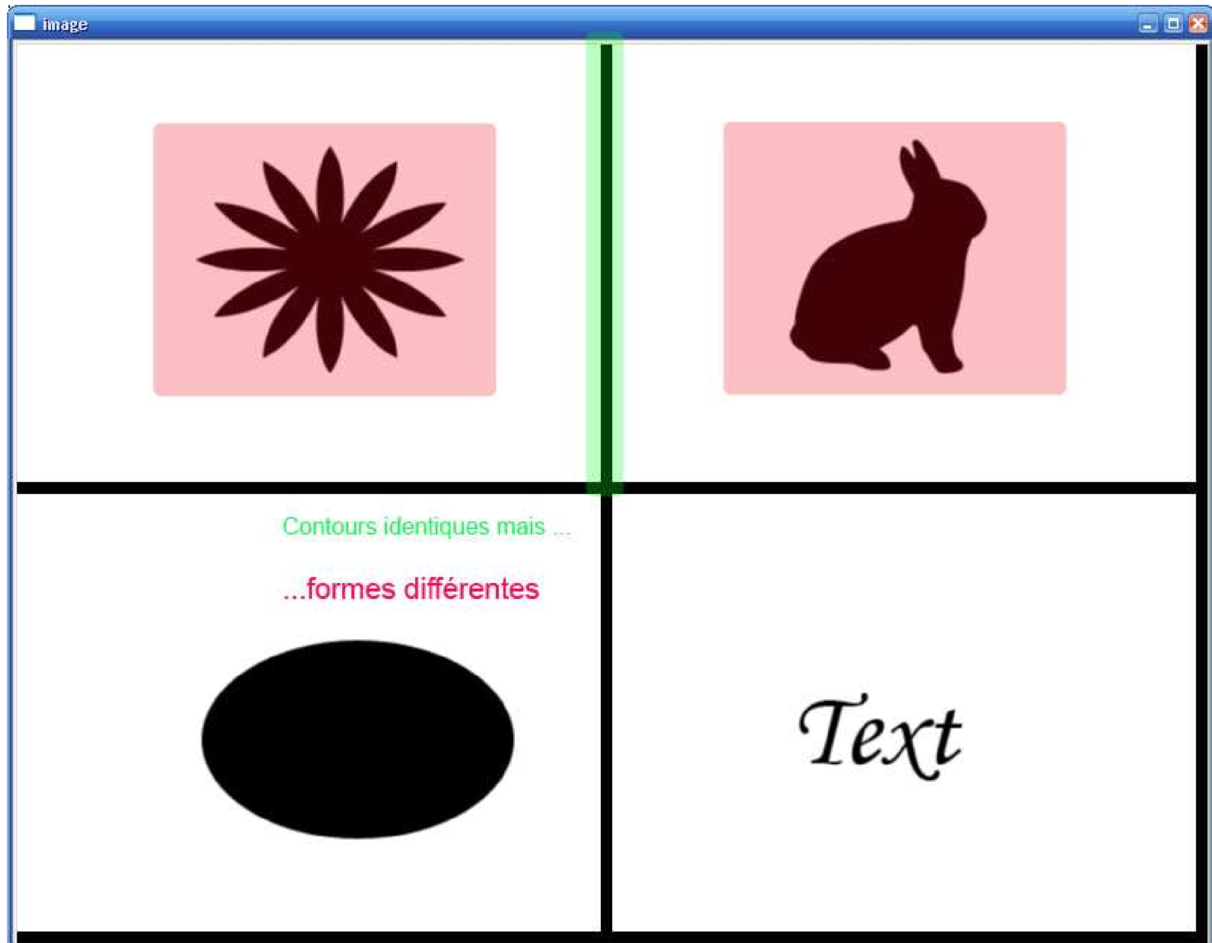
Nous avons alors utilisé la formule suivante pour calculer la distance entre deux pixels :

$$\Delta E^* = \sqrt{((L_1 - L_2)^2 + (a_1 - a_2)^2 + (b_1 - b_2)^2}$$

Avec : L_1, a_1, b_1 coordonnées dans l'espace colorimétrique CIELab du premier pixel et L_2, a_2, b_2 celles de la seconde.

3. Comparaison par formes

Nous nous sommes rendus compte par la suite que la comparaison par couleur ne suffisait pas. En effet pour certains cas, les couleurs correspondent bien pour les bords haut et bas de la pièce à rechercher mais l'intérieur de la pièce est complètement différent.



De cette manière, nous avons pensé qu'il serait bon d'implémenter une détection de formes. Pour cela, nous convertissons tout d'abord l'image d'origine en niveau de gris, puis nous appliquons une détection de contours en utilisant la fonction cvCanny. Nous avons décidé d'implémenter l'algorithme de Canny pour la détection de formes car il utilise plusieurs notions vues au cours de l'UV. Il applique tout d'abord un filtrage Gaussien afin de réduire le bruit sur l'image. Il va ensuite calculer l'intensité du gradient pour chacun des points de l'image. Une forte intensité indique une forte probabilité de présence d'un contour. Toutefois, cette intensité ne suffit pas à décider si un point correspond à un contour ou non. Seuls les points correspondant à des maxima locaux sont considérés comme correspondant à des contours, et sont conservés pour la prochaine étape de la détection.

Il va ensuite comparer cette intensité à deux seuils suivant la règle :

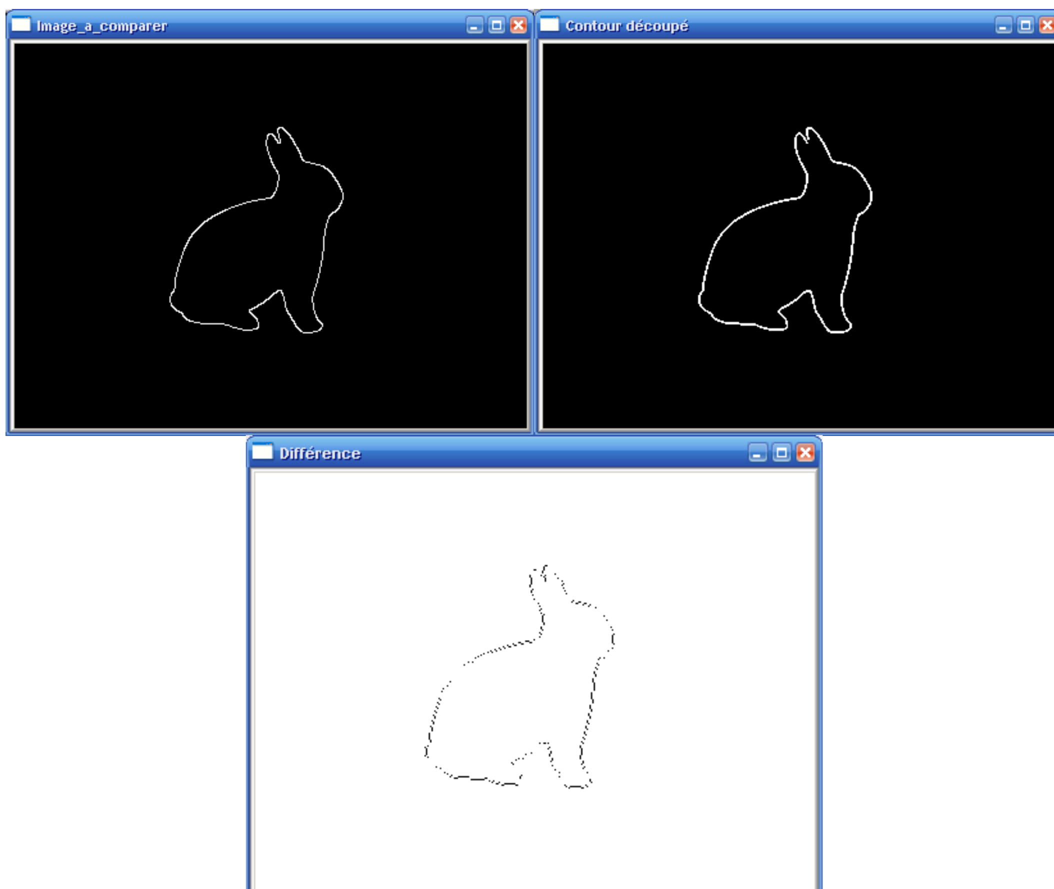
- Inférieur au seuil bas, le point est rejeté;
- Supérieur au seuil haut, le point est accepté comme formant un contour;
- Entre le seuil bas et le seuil haut, le point est accepté si il est connecté à un point déjà accepté.

Le seul problème de cette fonction est donc de déterminer les paramètres pour que ceux-ci puissent s'appliquer à chacune des images.

La détection de contours étant alors réalisée sur l'image d'origine, étant donné une pièce retournée suite à la comparaison sur les couleurs, nous allons convertir cette pièce en niveau de gris et lui appliquer une détection de contour.

Etant donné que l'on connaît la position de la pièce recherchée, on va découper dans notre image des contours, la pièce correspondante.

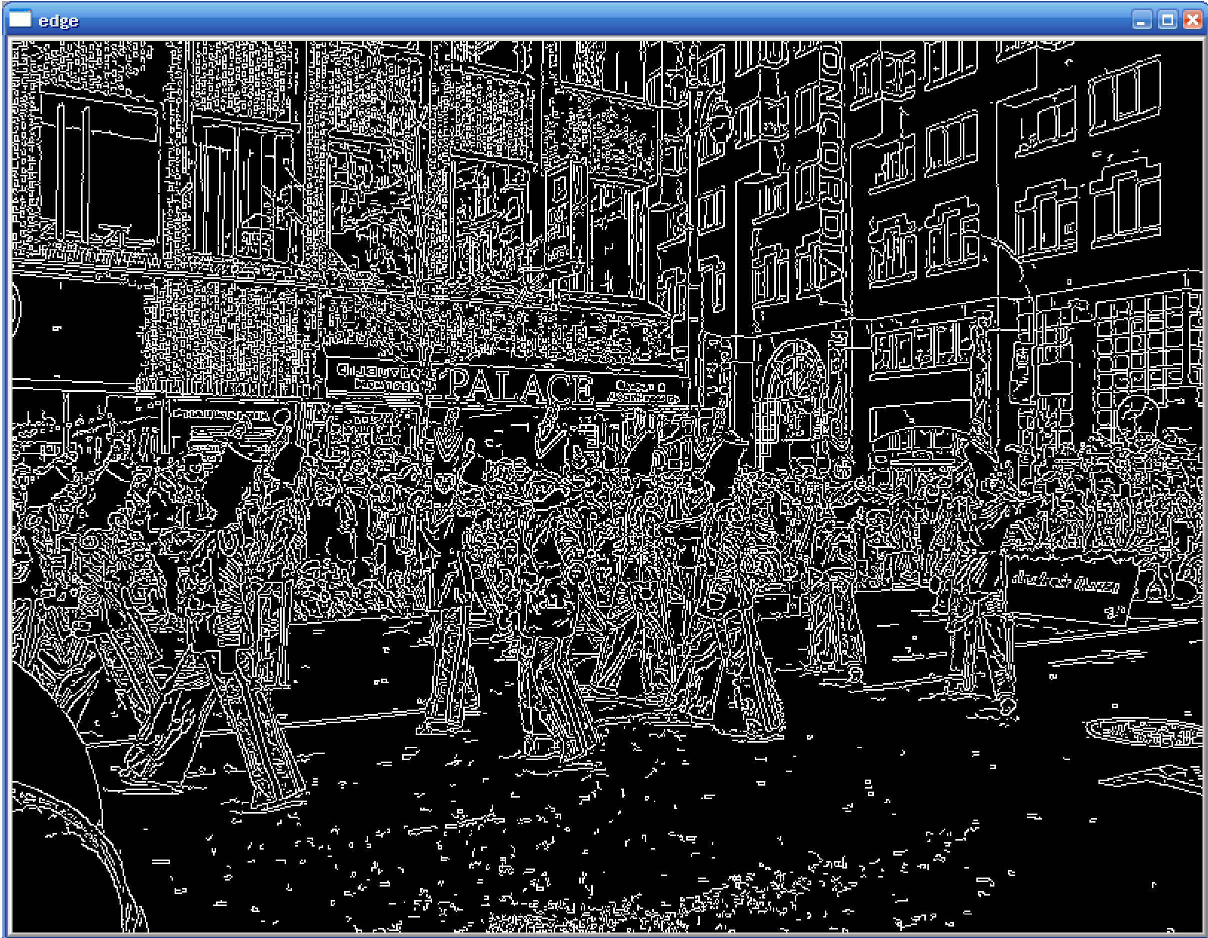
Ensuite, on va appliquer un seuillage sur chacune de ces deux images et faire alors une comparaison en utilisant la fonction cvCmp. On obtient alors une image présentant la différence entre les deux images, on calcule alors le nombre de pixels correspondant à cette différence et on le compare au nombre de pixels de l'image d'origine. S'il est inférieur, on a trouvé la bonne pièce sinon on enregistre le résultat et on recommence une comparaison de couleurs et ainsi de suite jusqu'à obtenir la bonne pièce.



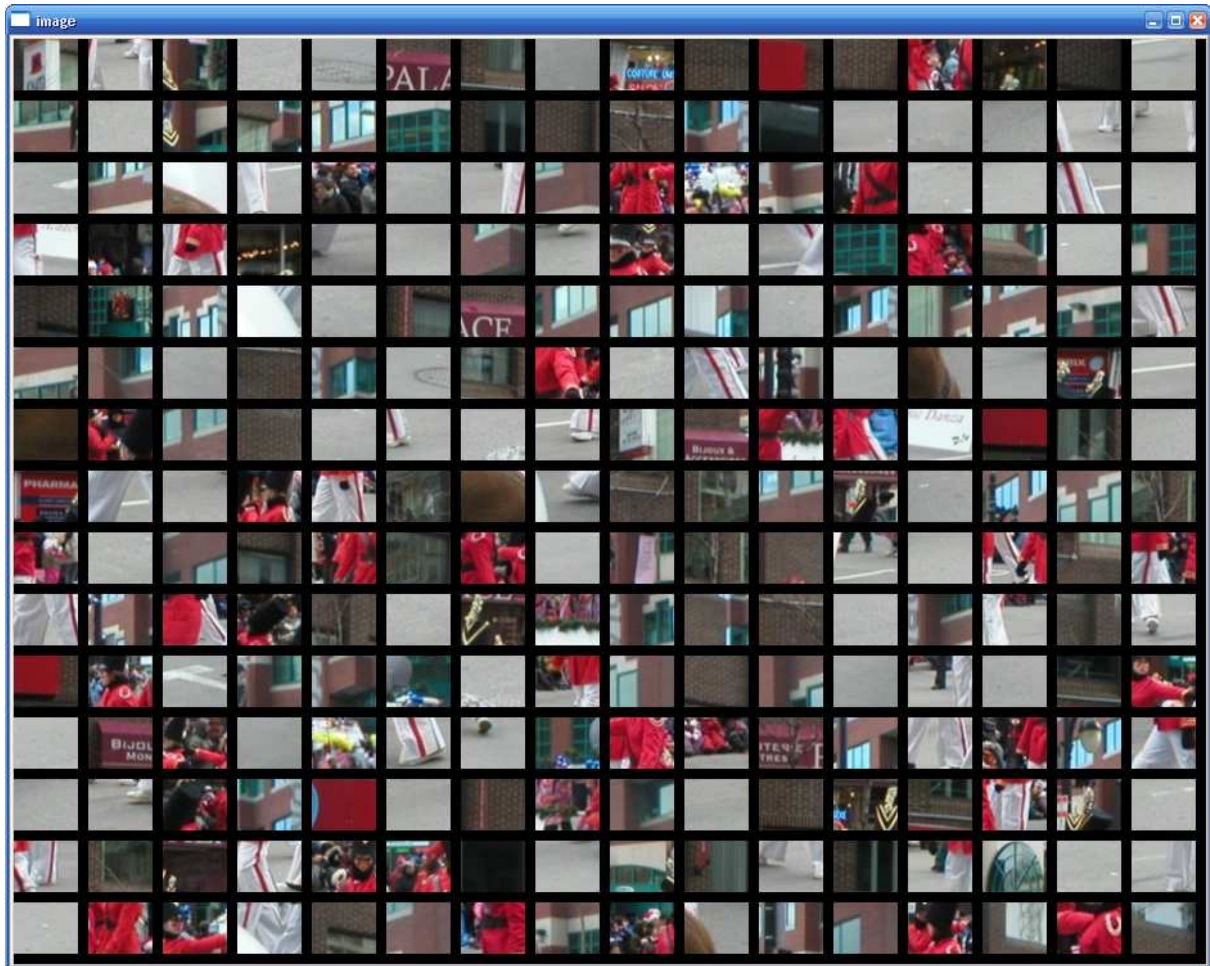
IV. Exemples

Image origine :

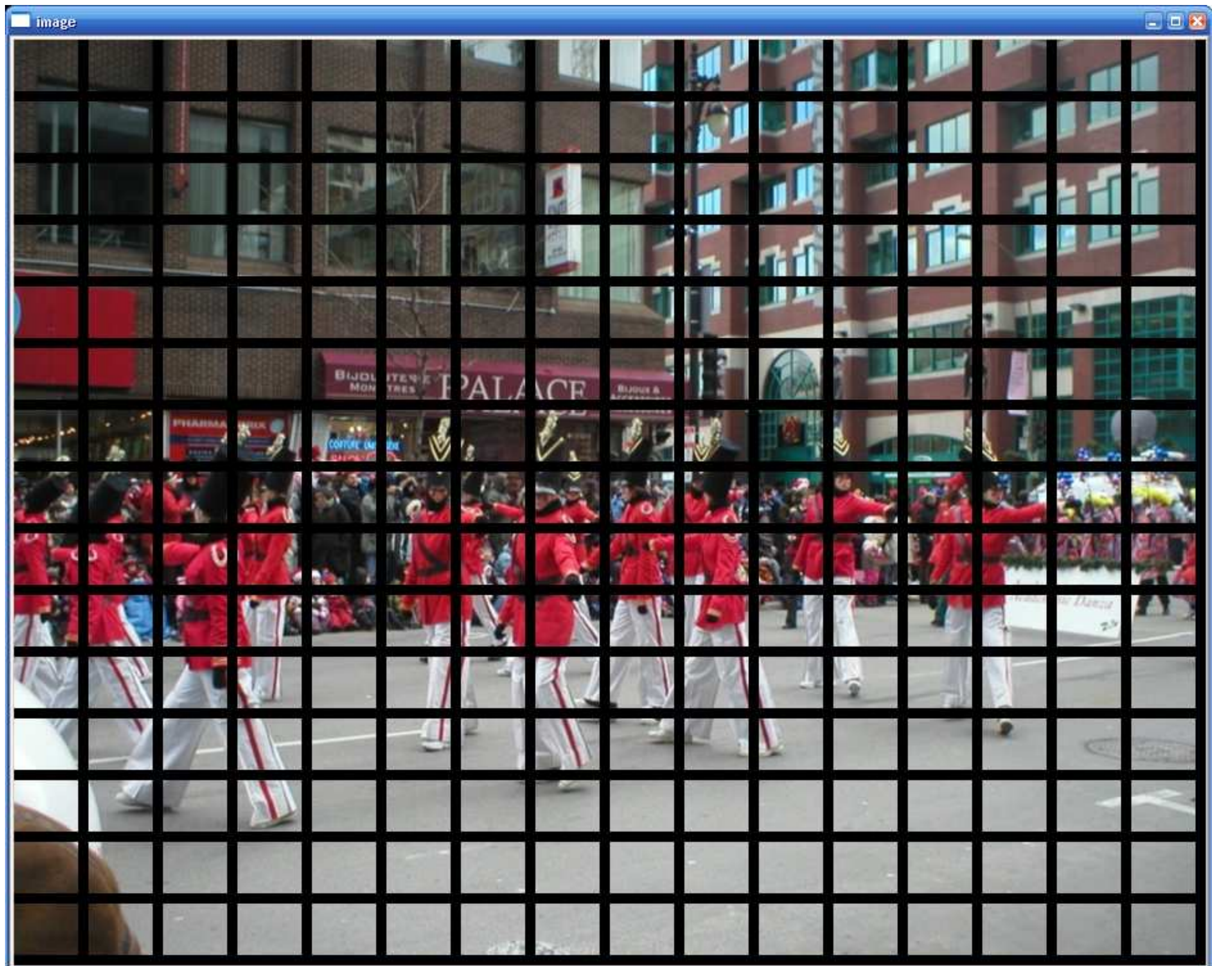
Contour image origine :



Puzzle :



Puzzle reconstitué :



CONCLUSION

Nous avons trouvé ce projet intéressant dans la mesure ou il nous as permis de mettre en application certaines notions vues au cours de ce semestre comme la détection de contour, le seuillage, …

Cependant, nous n'avons pas pu approfondir certains points, notamment la mise en place des seuils. Effectivement suivant le type de l'image, la détection de contours ne fournit pas le même résultat étant donné que les seuils appliqués au filtre sont les mêmes. Il aurait donc été intéressant de pouvoir adapter ces seuils suivant la nature de l'image.