Algorithmique et Programmation

—

Projets 2009/2010

—

G1: monasse*(at)*imagine.enpc.fr
G2: courchay*(at)*imagine.enpc.fr
G3: eric.bughin*(at)*gmail.com
G4: david.ok*(at)*imagine.enpc.fr
G5: francois.olivier.c.h*(at)*gmail.com
G6: vialette*(at)*univ-mlv.fr

1 Stéréo, Carte de Disparité et Reconstruction 3D

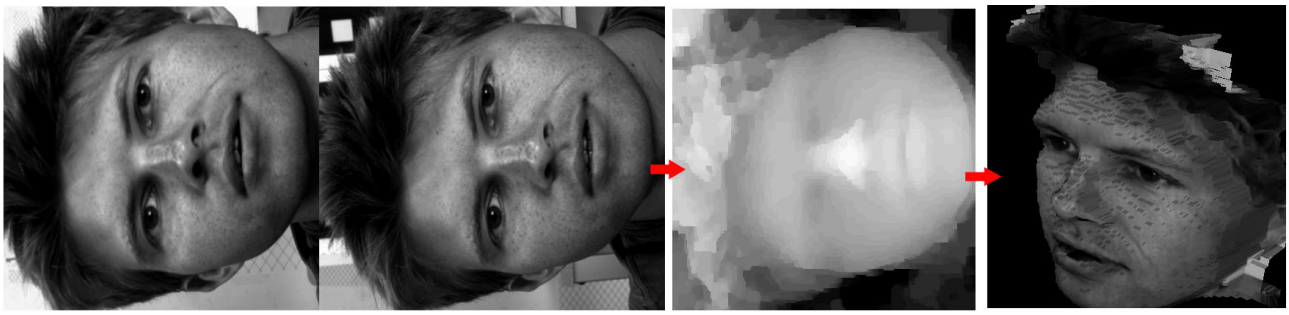


FIG. 1 – Deux images. Carte de disparité. Reconstruction 3D

1.1 Introduction

Lorsqu'on dispose de deux images d'un même objet, on parle de paire stéréo et il existe des techniques pour retrouver la troisième dimension et reconstruire un modèle 3D de l'objet. L'objectif de ce projet est de réaliser un programme capable de reconstruire automatiquement un objet 3D (un visage par exemple) à partir d'une paire stéréo. Nous allons réaliser ce projet par étapes successives, en ordre inverse par rapport à la chaîne de traitement complète.

1.2 Mise en correspondance

Nous supposons données deux images I_1 et I_2 telles que les points qui se correspondent sont sur les mêmes horizontales. On parle d'images rectifiées. Il faut retrouver la disparité $d(u,v)$ telle que le point $m_1 = (u_1, v_1)$ de I_1 corresponde au point $m_2 = (u_2, v_2)$ de I_2 avec $u_2 = u_1 + d(u1, v1)$ et $v_2 = v_1$.

On essaie de trouver le correspondant dans la deuxième image par corrélation. Pour un pixel (x_1, y_1) de I_1, on estime la "ressemblance" du pixel (x_2, y_2) de I_2. Pour cela, on compare les valeurs des pixels autour de (x_1, y_1) et de (x_2, y_2). Plus précisément, on mesure la corrélation entre deux "fenêtres" autour de ces points. On définit par étapes la corrélation C_{12} :

– Moyenne :

$$\bar{I}_i(x_i, y_i) = \frac{1}{(2N+1)^2} \sum_{u=-N}^{N} \sum_{v=-N}^{N} I_i(x_i + u, y_i + v)$$

– Produit scalaire :

$$\langle I_i, I_j \rangle (x_i, y_i, x_j, y_j) = \frac{1}{(2N+1)^2} \sum_{u=-N}^{N} \sum_{v=-N}^{N} (I_i(x_i + u, y_i + v) - \bar{I}_i(x_i, y_i)) \\ (I_j(x_j + u, y_j + v) - \bar{I}_j(x_j, y_j))$$

– Norme :

$$|I_i|(x_i, y_i) = \sqrt{\langle I_i, I_i \rangle (x_i, y_i, x_i, y_i)}$$

– Corrélation-croisée normalisée :

$$C_{12}(x_1, y_1, x_2, y_2) = \frac{\langle I_1, I_2 \rangle (x_1, y_1, x_2, y_2)}{|I_1|(x_1, y_1)|I_2|(x_2, y_2)}$$

qui vérifie $-1 \leq C_{12} \leq 1$ (ce qui est un bon test de correction du programme !)

Pour un point (x_1, y_1), on pourrait choisir alors comme point correspondant le point (x_2, y_2) maximisant la corrélation $C_{12}(x_1, y_1, x_2, y_2)$. Dans notre cas la recherche se réduit le long d'une ligne et on cherche donc à maximiser $C_{12}(x_1, y, x_2, y)$ où y est fixé par le pixel de I_1.

Commencez par programmer les deux fonctions suivantes :

1. La corrélation $C_{12}(x_1, y, x_2, y)$
2. La correspondance qui pour un pixel p_1 dans I_1 renvoie son pixel correspondant p_2 ainsi que la valeur de corrélation.

Des images rectifiées sont fournies avec le projet.

1.3 Carte de disparités

L'objectif est à présent de calculer la disparité et la pofondeur pour tous les pixels de l'image. Calculer la disparité des pixels dans l'ordre ne permet pas d'obtenir des resultats convenables. Il faut d'abord commencer par les points pour lesquels la disparité est fiable et propager le calcul aux voisins en utilisant les disparités déjà connues.

On appelle graines les points qui ont le meilleur taux de corrélation et donc pour lesquels la valeur de la disparité sera d'autant plus fiable. Ces points sont ceux dont la corrélation dépasse un certain seuil que vous déterminerez par expérimentation.

On procède de la manière suivante :

1. Pour chaque graine $g_i = p_1^i$ dans I_1, on calcule le pixel p_2^i correspondant dans I_2, la disparité est notée $d_i = p_2^i - p_1^i$. On note $\mathcal{V}(g_i)$ l'ensemble des pixels voisins de g_i.
2. Pour chaque pixel $p_1^k \in \mathcal{V}(g_i)$, le correspondant p_2^k dans I_2 est tel que

$$p_2^k(x) = p_2^i(x) + (p_1^k(x) - p_1^i(x)) + l$$

où l est choisi pour maximiser la corrélation entre p_1^k et p_2^k, avec $|l| <= v$. En d'autre termes la variation de disparité est bornée : $|d_i - d_k| <= v$ où d_k est la disparité du pixel p_k (prendre $v = 1$ pixel).

3. Ajouter p_1^k dans la liste des graines (en queue !)
4. Itérer tant que la liste des graines n'est pas vide

Il est judicieux de traiter les graines dans l'ordre de leur taux de corrélation.

1.4 Visualisation 3D

La disparité est inversement proportionnelle à la profondeur ; représenter le visage fourni en 3D. Imagine++ vous permet d'entrer un modèle 3D et de laisser l'utilisateur tourner autour.

1.5 Votre visage en 3D !

Pour rectifier des images, il faut considérer la matrice F, dite fondamentale, de taille 3×3, reliant deux images stéréo et telle que ${}^t[m_2, 1]F[m_1, 1] = 0$ pour tout couple de points correspondants (m_1, m_2). L'utilisation de F pour la rectification est ici hors sujet. Un programme utilisant F pour calculer les images rectifiées est donné. Le calcul de F se fait en mettant en correspondance des points d'intérêt (cf. projet panorama) et en utilisant un algorithme RANSAC pour éliminer les correspondances fausses. Ces briques mises en place, vous pouvez reconstruire votre visage en 3D en prenant deux photos devant un coin de mur bien texturé.

2 Ray Tracing

2.1 Introduction

Le *ray tracing*, ou *lancer de rayons*, est une technique permettant de générer des images de synthèse. Cette méthode cherche à simuler le parcours inverse de la lumière de la scène vers l'oeil, et permet de gérer réflexions et réfractions de cette lumière sur les objets présents dans la scène.

FIG. 2 – Ray tracing

2.2 Principe

Pour chaque pixel de l'image à générer, un rayon est lancé du point de vue vers la scène. Lors d'une intersection avec un objet de la scène, la trajectoire de ce rayon est réfléchie ou réfractée (suivant les propriétés de l'objet), le rayon continue sa course, et ainsi de suite jusqu'à rencontrer un objet non réfléchissant et non réfractant, ou jusqu'à un nombre défini de réflection/réfraction. Chacun de ces impacts apportera une contribution à la couleur apparente du pixel courant, suivant des coefficients de réflexion/réfraction.

Pour chacun de ces impacts, pour déterminer la luminosité en ces points, un rayon est lancé depuis chaque source lumineuse. Combinée à la couleur propre de l'objet, on peut ainsi déterminer la couleur finale en ce point. Un schéma est présenté en figure 3.

Pour le calcul de l'illumination d'un point, on se basera sur le *modèle d'illumination de Phong*, décrit ici : http://fr.wikipedia.org/wiki/Ombrage_Phong. On ne s'intéressera pas à la dernière partie, l'interpolation de Phong, non cohérente avec le lancer de rayon. Pour les différents modèles d'illumination existants, voir http://ph.ris.free.fr/these/, chapitre 2.

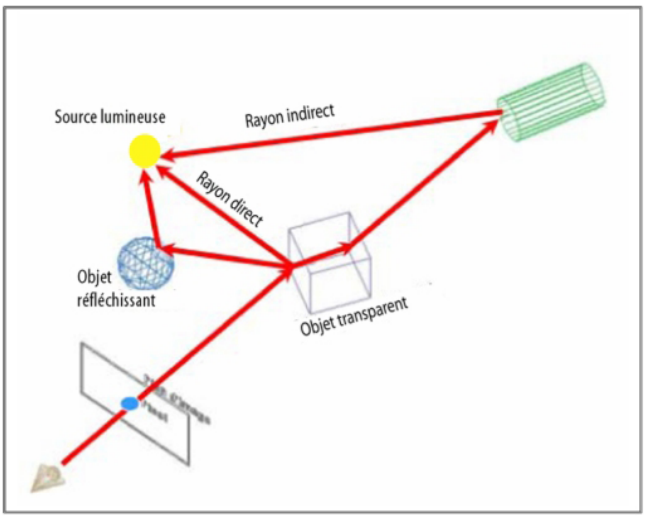


FIG. 3 – Trajectoire d'un rayon

Pour la construction des objets 3D à visualiser, on se basera sur les *CSG*, ou *Constructive Solid Geometry*. Le principe est de construire des objets plus ou moins complexes en faisant des opérations simples sur des objets de bases : union, intersection, différence (pour de plus amples détails ainsi que des exemples en image, voir http://en.wikipedia.org/wiki/Constructive_solid_geometry).

2.3 A faire au minimum

1. Comprendre le principe du ray tracing (voir http://fr.wikipedia.org/wiki/Lancer_de_rayon en particulier) ;
2. Commencer par un programme simple ne gérant que des sphères affichées uniformément (pas de notion de source lumineuse, seulement le terme ambiant du modèle de Phong) ;
3. Rajouter la gestion d'une autre primitive géométrique : le plan orienté, définissant un demi espace ;
4. Utiliser les *CSG* pour construire de nouveaux objets. On pourra par exemple construire un cube de façon simple en utilisant des plans orientés ou encore tester des objets plus avancés (exemple : un dé peut-être construit par différences entre un cube et plusieurs sphères, etc.) ;
5. Rajouter la gestion des occlusions : un objet n'est éclairé par une source lumineuse que s'il n'existe aucun autre objet entre lui et cette source ;
6. Rajouter des sources lumineuses ponctuelles, sans gestion de réflection ni réfraction : chaque sphère aura un aspect mat (on ajoute le terme de diffusion du modèle de Phong). On se se préoccupera pas non plus des occlusions éventuelles.
7. Rajouter le terme spéculaire dans le modèle d'illumination.

2.4 Extensions possibles

1. Rajouter un affichage progressif : ne pas calculer les pixels ligne par ligne mais par grilles de plus en plus fines, en les affichant au fur et à mesure ;
2. Placer la description des scènes dans un fichier annexe : votre programme principal ne comportera alors que le "'moteur de rendu"' et se contentera d'interpréter le fichier de données ;
3. Rajouter la gestion des reflets.
4. Optimiser le calcul d'intersections avec une notion de boites englobantes ;
5. ...

2.5 Quelques liens en plus

– Support de cours sur le *ray tracing* :
http://www.cs.virginia.edu/~gfx/courses/2004/Intro.Fall.04/handouts/05-raycast.pdf ;
– Explications et exemples sur le modèle d'illumination de Phong (et d'autres) :
http://artis.imag.fr/~Nicolas.Holzschuch/cours/CIV01/CIV01_materials_small.pdf,
http://www.cs.virginia.edu/~gfx/courses/2004/Intro.Fall.04/handouts/06-light.pdf ;

3 Construction de Panorama par Ransac

3.1 Introduction

Si on a un ensemble d'images prises par une camera tournante les images sont liées les unes aux autres par une transformation simple. On va donc facilement pouvoir transformer chaque image vers l'espace d'une image de référence et on obtiendra ainsi une image unique offrant un très grand champs angulaire comme montré en Figure 4.

Lorsque deux cameras sont uniquement en rotation l'une par rapport à l'autre et sans translation (le centre optique ne bouge pas) alors la coordonnée d'un point x dans l'image 1 visualisant un point $3D$ P de la scène, se déduit du point x' dans l'image 2 correspondant au même poit P. On a en fait $x' = Hx$. Ainsi pour une série d'image prise par une caméra tournante on en déduit à-partir des mises en correspondance de points des mises en correspondance d'images grâce aux homographies H. De cette façon on va pouvoir rectifier toutes les images pour les mettre en correspondance avec une image de référence choisie (à-priori l'image centrale en terme de point de vue angulaire dans la série de photos) et ainsi reconstruire un panorama.

3.2 Un peu de Théorie : paire de cameras en rotation

Nous supposons que la camera est un modèle parfait pinhole, dans ce cas dans le repère attaché à la camera on a une transformation simple qui est représenté sur la Figure 5.

En utilisant le théorème de thalès on obtient :

$$\begin{cases} x' &= \frac{f'x}{z} \\ y' &= \frac{f'x}{z} \end{cases}$$

FIG. 4 – Exemple de reconstitution de panorama

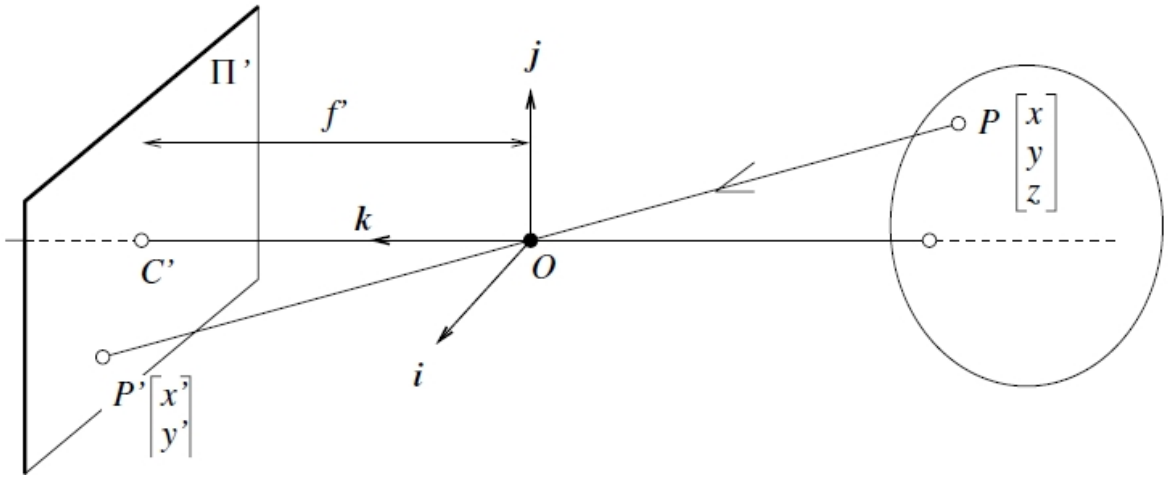


FIG. 5 – Modéle de caméra pinhole

Si le centre dans le plan image est dans le coin en haut à gauche de l'image comme c'est souvent le cas on obtient alors :

$$\begin{cases} x' &= \frac{f'x}{z} + u_0 \\ y' &= \frac{f'x}{z} + v_0 \end{cases}$$

Si nous utilisons le système de coordonnées projective, ie si nous rajoutons une troisième coordonnée nous avons :

$$\begin{pmatrix} x'Z \\ y'Z \\ Z \end{pmatrix} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f' & 0 & u_0 \\ 0 & f' & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = K \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Maintenant si le repère est toujours centré au centre optique de la caméra mais avec une rotation on obtient :

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = KR \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

et pour la seconde caméra qui à le même centre optique mais une autre rotation on a :

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = K'R' \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

les matrices K et R sont inversibles on en déduit :

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = KRR'^{-1}K'^{-1} \begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = H \begin{pmatrix} u' \\ v' \\ w' \end{pmatrix}$$

pour repasser des coordonnées projectives aux coordonnées euclidiennes il suffit de diviser par la troisième coordonnée w soit $x = \frac{u}{w}$ et $y = \frac{v}{w}$. La matrice H à 9 coefficients et peut-être multipliée par un réel λ quelconque. Donc on a 8 coefficient indépendants et on en déduit la relation homographique en développant la relation matricielle ci-dessus et en divisant par la troisième coordonnée : $x = \frac{a_1x'+a_2y'+a_3}{a_7x'+a_8y'+1}$ et $y = \frac{a_4x'+a_5y'+a_6}{a_7x'+a_8y'+1}$
Ainsi avec les coordonnées de 1 point trouvez la relation du type :

$$A_1 \underbrace{\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix}}_{h} = b_1$$

En déduire qu'avec 4 points on a la relation

$$\underbrace{A}_{8\times8} h = b$$

Ainsi on peut simplement en déduire les 8 coefficients de l'homographies. Une fois cette homographie connue on peut facilement en déduire les coordonnées théoriques dans l'image 1 et les comparer aux coordonnées réelles pour d'autres correspondances que celles utilisées pour calculer la relation homographique :

$$\begin{cases} x_{theory} &= \frac{a_1x'+a_2y'+a_3}{a_7x'+a_8y'+1} \\ y_{theory} &= \frac{a_4x'+a_5y'+a_6}{a_7x'+a_8y'+1} \end{cases}$$

3.3 Calcul de l'Homographie par RANSAC

La première étape consiste à extraire des points d'intérets dans les images et à les mettre en correspondance entre 2 images. Etant donner un point d'intéret dans une image on cherche tous les descripteurs des points d'intérets dans l'autre image et on conserve le plus proche comme correspondance si il est suffisemment proche on le considère comme une correspondance. Le programme extrayant les points d'intérets, les descripteurs et la mise en correspondance vous est fourni avec un fichier *test.cpp* pour avoir un exemple d'utilisation.

Dans cet ensemble E de correspondances on a des correspondances effectivement correctes *inliers* et des correspondances fausses dites *outliers*. On cherche a déterminer les inliers et les outlier grâce à un algorithme, le Ransac, que vous allez implémenter.

Le RANSAC (RANdom SAmple Consensus) pour simplifier consiste à itérer un grand nombre de fois N. A chaque itération on récupère s correspondances au hasard nécessaires à l'estimation du modèle, parmi l'ensemble de correspondances E. ici le $s = 4$ et le modèle est l'homographie h. Ensuite on teste le nombre de points en adéquation avec le modèle, soit le nombre d'inlier n_i. Ici un point est un inlier si l'erreur de reprojection est sous un seuil $treshold$ (en général $treshold = 1$ pixel) :

$$\begin{cases} x1_{theory} = \frac{a_1x_2+a_2y_2+a_3}{a_7x_2+a_8y_2+1} \\ y1_{theory} = \frac{a_4x_2+a_5y_2+a_6}{a_7x_2+a_8y_2+1} \\ \text{si } (x_1 - x1_{theory})^2 + (y_1 - y1_{theory})^2 < treshold^2 \text{ alors c'est un inlier} \end{cases}$$

à la fin de chaque itération on test si on a plus d'inliers que pour les itérations précédentes si c'est le cas on enregistre et met à jour le plus grand nombre d'inlier et la meilleure homographie, n_{ibest} et h_{best} ainsi que un tableau d'indice des inliers $Ind_{inliers}$. Au bout des N itérations on à le modèle h_{best} qui a permis de retrouver le plus grand nombre d'inliers ainsi que les indices des inliers $Ind_{inliers}$.

A-partir de là après le Ransac on estime la meilleure solution avec l'ensemble des inliers, notre équation vu pour estimer le modèle à-partir de 8 points devient :

$$\underbrace{A}_{(2*ninliers)\times 8} \quad h = \quad \underbrace{b}_{(2*ninliers)}$$

La solution approchée est :

$$\underbrace{A}_{(2*ninliers)\times 8} \quad h_{final} = A.\text{PseudoInverse} \quad \underbrace{b}_{(2*ninliers)}$$

Pour estimer le modèle final avec les $ninliers$ points il est préférable de donner une égale importance à chaque ligne de l'équation $Ah = b$ pour celà on peut simplement diviser chaque ligne de A par sa norme et diviser les coefficients de b en adéquation. Il est préférable d'utiliser une normalisation plus adaptée au problème qui consiste à travailler sur des points normalisés tels que les inliers aient pour moyenne 0 et une déviation standard de 1. On à alors les équations matricielles utilisant les coordonnées projectives (u, v, w) (on note avec un ˆ les données dans l'espace normalisé) :

$$\begin{pmatrix} \hat{u}_{inlier1} \\ \hat{v}_{inlier1} \\ \hat{w}_{inlier1} \end{pmatrix} = \underbrace{Norm_1}_{3\times 3} \begin{pmatrix} u_{inlier1} \\ v_{inlier1} \\ w_{inlier1} \end{pmatrix}$$

$$\begin{pmatrix} \hat{u}_{inlier2} \\ \hat{v}_{inlier2} \\ \hat{w}_{inlier2} \end{pmatrix} = \underbrace{Norm_2}_{3\times 3} \begin{pmatrix} u_{inlier2} \\ v_{inlier2} \\ w_{inlier2} \end{pmatrix}$$

$$\begin{pmatrix} \hat{u}_{inlier1} \\ \hat{v}_{inlier1} \\ \hat{w}_{inlier1} \end{pmatrix} = \hat{H} \begin{pmatrix} \hat{u}_{inlier2} \\ \hat{v}_{inlier2} \\ \hat{w}_{inlier2} \end{pmatrix}$$

En déduire la fonction de dénormalisation permettant de passer de $\hat{H}$ à H, en fonction de $Norm_1$ et $Norm_2$.

3.4 Démarche générale et Implémentation

On calculera les homograhies de proche en proche entre 2 images consécutives. Puis on calculera les composition d'homographies qui mermettent de passer d'une image à l'image centrale de référence.Pour chaque image on calculera la *bouding box* en fonction des coordonnées des 4 sommets d'une image et de leur transformations homographiques dans l'image de référence.Ainsi avec toutes les images on obtiendra la taille minimale de la fenêtre qu'il nous faut.

Du point de vue de l'implémentation un programme de test utilisant les librairies nécessaires et leur utilisation vous est fourni. principalement : calcul matriciel, extraction de points d'intérets, mise en correspondance. pour calculer les matrices inverses et pseudo-inverses on utilise An = A.inverse() et An = A.pinverse() pour utiliser ces fonctionnalités elles doivent être du type *Matrix* mais pour les stocker dans en mémoire dans le ransac on les transferrera vers des *SmallMatrix* qui prennent moins de place.

Vous pouvez commencer par utiliser les images fournies avec le projet puis chercher sur le web d'autres images permettant de reconstituer des panoramas.

3.5 Travail Supplémentaire : Recollage d'Image

Lorsque vous recollez les images vous pouvez voir nettement les frontières de recollement, car les images A et B ont des différences d'intensité lumineuse. Si il vous reste du temps, Il est donc plus élégant de faire une frontière progressive. Le problème est de recoller deux images A et B en une image C sans que cela ne se voit. On part de deux méthodes.

L'idée la plus naïve consiste à construire $C(x) = (1 - \lambda)A(x) + \lambda B(x)$ avec λ qui passe progressivement de 0 à 1.

Si il vous reste d'avantage de temps vous pouvez essayer la méthode suivante qui est encore plus efficace : Le principe du mélange multi-échelle est de faire varier λ de 0 à 1 sur une bande de largeur dépendant de la fréquence. Plus exactement, on décompose les images en fréquence. On mélange leur spectre sur une largeur différente pour chaque fréquence. Puis on revient à l'espace initial pour obtenir l'image finale. Cette méthode est décrite ici
http://web.mit.edu/persci/people/adelson/pub_pdfs/pyramid83.pdf
et là
http://web.mit.edu/persci/people/adelson/pub_pdfs/spline83.pdf

4 Dames

4.1 Objectif

Il s'agit d'écrire un programme jouant aux Dames selon les règles. Le programme doit être le meilleur possible. Vous utiliserez pour cela l'algorithme $\alpha - \beta$ de recherche du meilleur coup à jouer (Cf. document joint).

4.2 Règles du jeu

Les règles sont disponibles sur le site de la Fédération Française de Jeu de Dames (http://www.ffjd.fr/Web/index.php?page=reglesdujeu). On prendra garde de ne pas oublier les règles sur la prise obligatoire de pions (pour ceux qui ne la connaissent pas se référer au site).

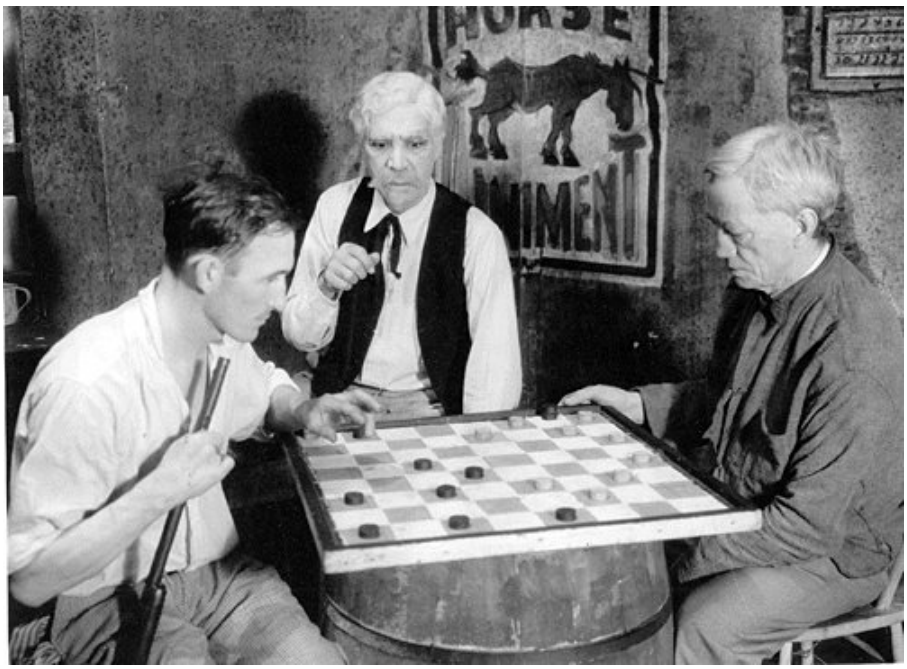


FIG. 6 – Une partie de Dames.

4.3 Résultats demandés

Vous devrez parvenir à écrire un programme qui joue le mieux possible aux Dames en respectant les règles, est capable de jouer avec les noirs ou les blancs, dialogue avec l'utilisateur, sait quand la partie est terminée et désigne le vainqueur.

Pour cela, le travail peut être séparé en plusieurs composantes :

4.3.1 Interface graphique

Vous devrez concevoir une interface graphique qui au minimum doit :
- afficher le damier et les pions,
- permettre d'entrer à la souris les coups joués par l'adversaire de l'ordinateur,
- afficher diverses informations (indiquer que l'ordinateur « passe » s'il ne peut jouer, indiquer que la partie est terminée et désigner le vainqueur, afficher éventuellement le nombre de pions pour chacun des camps en cours de partie...)

Il sera important dès le début de bien réfléchir aux structures de données utilisées pour représenter une situation du jeu de Dames, et un coup de jeu.

4.3.2 Règles du jeu

L'interface de jeu permet à deux joueurs de s'affronter :
- créer la situation initiale du jeu (20 pions de chaque couleur placés sur les 4 premières rangées de chaque côté),
- vérifier que les coups proposés par les joueurs sont valides à la fois pour un pion et pour une dame,
- vérifier que les coups joués correspondent bien au coup de prise maximale à chaque fois,

– réaliser la fonction de transition qui fait passer d'une situation du jeu à une autre en respectant les règles (déplacer le pion, et supprimer les pions adversent qui ont éventuellement été pris).

4.3.3 Stratégie de l'ordinateur

Comme un certain nombre de jeux, les Dames présentent les caractéristiques suivantes :
– deux adversaires jouent à tour de rôle
– la situation globale du jeu est connue de chacun des joueurs
– la chance n'intervient pas.

Dans un tel cas, l'ordinateur pourrait de manière théorique calculer la totalité de l'arbres des situations de jeu possibles, de manière à adopter la meilleure stratégie possible.

Evidemment, la taille de cet arbre est sans commune mesure avec les possibilités de calcul d'un ordinateur, et donc il n'est pas possible de faire jouer l'ordinateur selon cette meilleure stratégie possible (heureusement d'ailleurs, sinon le jeu ne présenterait pas beaucoup d'intérêt !). Remarquez que pour le jeu du morpion, on peut par contre dessiner cet arbre à la main, et qu'il existe une stratégie optimale qui permet à tous les coups de faire au moins match nul.

En pratique, l'ordinateur regarde toutes les possibilités un petit nombre de coups à l'avance, n (il « parcourt l'arbre du jeu jusqu'à la profondeur n »). Il décidera de jouer le coup qui amène le jeu dans la branche de l'arbre qui lui semble la plus favorable. Pour ce faire, il a besoin d'évaluer les situation du jeu n coups plus loin, en leur donnant une note selon que ces situations sont plus ou moins avantageuses pour lui ou pour son adversaire.

La programmation de la stratégie de l'ordinateur consistera donc en :
– l'algorithme `Minimax` qui réalise le parcours dans l'arbre du jeu, et son amélioration, l'algorithme $\alpha - \beta$
– une fonction d'évaluation du jeu

4.4 Algorithmes Minimax et $\alpha - \beta$

4.4.1 Justification

Les algorithmes de recherche de type Minimax (voir § 4.4.2), dont $\alpha - \beta$ (voir § 4.4.3) est une amélioration, s'appliquent à une catégorie de jeux à deux joueurs, qui présentent les caractéristiques suivantes :
– les deux adversaires jouent à tour de rôle,
– la situation globale du jeu est connue de chacun des joueurs,
– la chance n'intervient pas,
– ils sont dits « à somme nulle » : les gains d'un joueur représentent exactement les pertes de l'autre joueur.

Pour ces jeux, on peut définir une notion d'arbre de jeux par
– S l'ensemble de toutes les situations possibles du jeu,
– $succ : S \longrightarrow Reg(S)$ l'application qui à une situation s associent l'ensemble des situations s' telles qu'il existe un coup régulier faisant passer de s à s'.

On suppose qu'il existe une fonction $h : S \longrightarrow \mathbf{Z}$ qui donne la valeur d'une situation du jeu (voir § 4.4.4). Cette fonction, appliquée à chacune des situations accessibles légalement à partir de la situation courante permet de décider si un coup est meilleur qu'un autre. Toutefois, il peut être intéressant d'examiner les situations de jeu plus profondément pour tenir compte des réponses possibles de l'adversaire. On appelle P cette profondeur de recherche. Plus P est grand, plus l'arbre de recherche est grand et donc long à explorer et meilleur (en principe) est le coup joué.

L'exploration complète d'un arbre de jeu n'est généralement pas possible car ce serait trop long. C'est pourquoi on se limite à une profondeur P assez faible.

4.4.2 L'algorithme Minimax

Dans l'approche Minimax, les joueurs sont appelés respectivement Max et Min. Max choisit parmi tous les coups réguliers le coup qui a la valeur maximale tandis que Min choisit celui qui minimise le gain de Max. Dans un arbre Minimax, la racine (profondeur 0) représente la situation du jeu actuelle. L'évaluation avec la fonction h se fait à la profondeur P. L'algorithme Minimax (Cf. Fig. 7) remonte la meilleure valeur jusqu'à la racine ce qui détermine le meilleur coup à jouer.

La figure 8 montre un arbre Minimax de profondeur 2.

Pour éviter d'avoir à faire des min et des max, on emploie une procédure Negamax (Cf. Fig. 9) qui ne fait que des max. Pour cela, il suffit de considérer que le joueur Min prend la valeur maximale des valeurs opposées des situations filles.

4.4.3 Amélioration du Minimax : $\alpha - \beta$

L'algorithme Minimax effectue une exploration complète de l'arbre de recherche jusqu'à un niveau donné. Il peut arriver qu'une exploration partielle de l'arbre suffise. Pour cela, il faut éviter d'examiner des sous-arbres qui ne conduiront pas à des situations dont la valeur contribuera au calcul du gain à la racine de l'arbre.

```
Minimax (s : situation)
  m, t : entier ;
  Si s est une feuille
    Alors renvoyer h(s) ;
  Si s est de type Max Alors
    m ← −∞ ;
    Pout tout s′ : situation ∈ succ(s) Faire
      t ← Minimax(s′) ;
      Si t > m Alors m ← t ;
    Fin Pour tout
    renvoyer m ;
  Si s est de type Min Alors
    m ← +∞ ;
    Pout tout s′ : situation ∈ succ(s) Faire
      t ← Minimax(s′) ;
      Si t < m Alors m ← t ;
    Fin Pour tout
    renvoyer m ;
Fin Minimax
```

FIG. 7 – Procédure Minimax

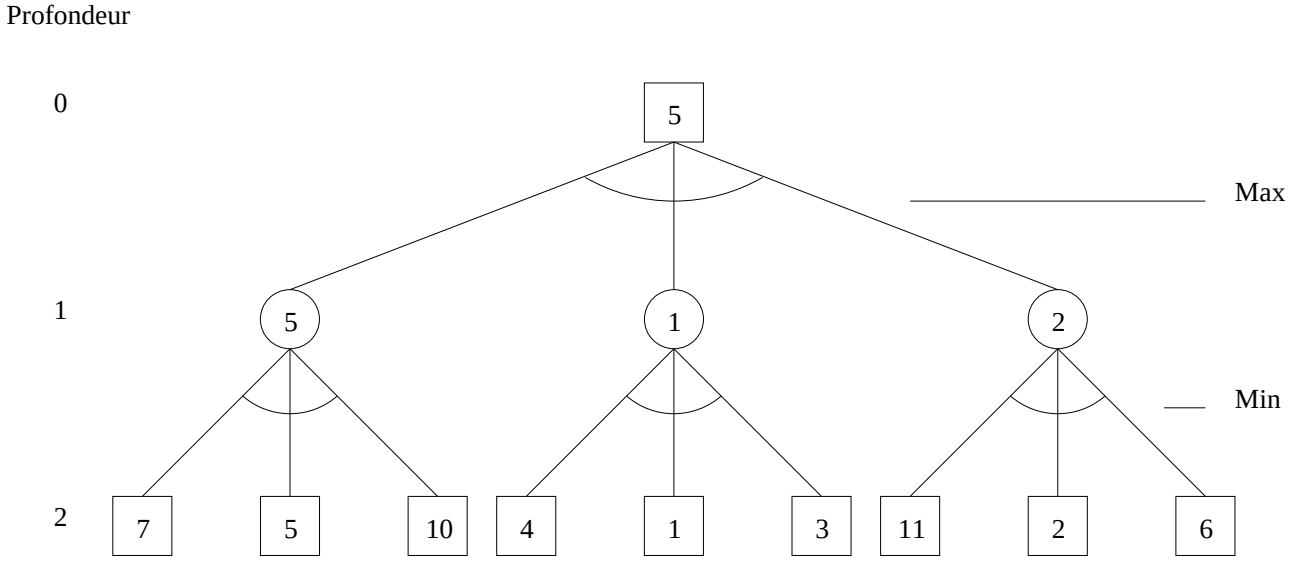


FIG. 8 – Arbre Minimax de profondeur 2

L'arbre de recherche est exploré en profondeur d'abord par l'algorithme, l'ordre de visite des nœuds est donc totalement fixé. Sur la figure 10 les branches de gauche sont parcourus en premier.

Dans les exemples de la figure 10 certains nœuds ont une valeur définitive alors que les autres (étiquetés avec un nom de variable) n'en ont pas encore reçue. D'après la définition de l'algorithme Minimax la valeur de la situation racine de l'arbre *(a)* est obtenue par

$$u = \max\{5, v\}, \text{ où } v = \min\{4, \ldots\}.$$

Il est clair que $u = 5$ indépendamment de la valeur de v. Il en résulte que l'exploration des branches filles du nœud étiqueté par v peut être arrêtée. On dit qu'on a réalisé une *coupure superficielle*. En appliquant récursivement le même raisonnement à l'arbre *(b)*, on en déduit que la valeur de u peut être obtenue sans connaître la valeur finale de y. De même que précédemment, l'exploration des branches filles du nœud étiqueté par y n'est pas nécessaire : on parle alors de *coupure profonde*.

Plus généralement, lorsque dans le parcours de l'arbre de jeu par Minimax il y a modification de la valeur courante d'un nœud, si cette valeur franchit un certain seuil, il devient inutile d'explorer la descendance encore inexplorée de ce nœud.

On distingue deux seuils, appelés pour des raisons historiques, α (pour les nœuds Min) et β (pour les nœuds Max) :

```
Negamax (s : situation)
  m, t : entier ;
  Si s est une feuille
    Alors renvoyer h(s) ;
  m ← −∞ ;
  Pout tout s' : situation ∈ succ(s) Faire
    t ← −Negamax(s') ;
    Si t > m Alors m ← t ;
  Fin Pour tout
  renvoyer m ;
Fin Negamax
```

FIG. 9 – Procédure Negamax

- le seuil α, pour un nœud Min s, est égal à la plus grande valeur (déjà déterminée) de tous les nœuds Max ancêtres de s. Si la valeur de s devient inférieure ou égale à α, l'exploration de sa descendance peut être arrêtée ;
- le seuil β, pour un nœud Max s, est égal à la plus petite valeur (déjà déterminée) de tous les nœuds Min ancêtres de s. Si la valeur de s devient supérieure ou égale à β, l'exploration de sa descendance peut être arrêtée.

On donne figure 11 l'énoncé de l'algorithme $\alpha - \beta$, qui maintient ces deux seuils pendant le parcours de l'arbre. Au départ, la procédure Alphabeta est appelée avec un intervalle (α, β) égal à $]-\infty, +\infty[$.

4.4.4 Fonction d'évaluation

D'une façon générale, on donne à la fonction d'évaluation la forme suivante :

$$p_{mat}c_{mat} + p_{mob}c_{mob} + p_{pos}c_{pos}$$

qui prend en compte trois critères : le matériel (c_{mat}), la mobilité(c_{mob}) et la position (c_{pos}). Chaque critère est accompagné d'une pondération qui indique l'importance accordée à ce critère et qui peut être variable au cours d'une partie.

Remarque : Pour estimer la valeur globale d'une situation, on doit faire intervenir la différence entre l'évaluation de la situation pour un camp et l'évaluation de la situation pour l'autre camp.

4.5 Indications

4.5.1 Fonction d'évaluation

Pour le jeu de Dames, deux critères principaux d'évaluation :
- d'une part, éliminer le plus possible de pions adverses,
- d'autre part, pouvoir transformer ses pions en dames (qui présentent plus de possibilités qu'un simple pion).

4.5.2 Stratégie

Dans les programmes classiques de jeux de dames, on distingue plusieurs phases de jeu.
- L'ouverture (resp. la fermeture), qui représente environ les douze premiers (resp. derniers) coups. On se sert habituellement de parties déjà existantes pour savoir quel va être le meilleur coup à jouer.
- Le milieu de partie, où l'objectif va être de prendre le plus de pions possible ainsi que de créer le plus grand nombre de dames possible (on se sert ici de l'algorithme alpha-beta).

Dans notre cas, on utilisera l'algorithme alpha-beta dans tous les cas de figures (sauf si vous connaissez certaines tactiques de jeux pour les Dames). La fonction d'évaluation se limitera donc à une somme de deux termes : le potentiel de pions pris et le nombre de dames créées. Ces deux termes correspondent à la partie matérielle de la fonction d'évaluation mais sont amplement suffisants pour les dames.

4.5.3 Jeu

Pour éviter que l'ordinateur joue exactement les mêmes coups sur une même partie, on peut ajouter un peu d'aléatoire dans les choix qu'il fait.

Au terme du projet, on pourra faire s'affronter en duels les algorithmes des différents binômes (avec un temps de jeu limité !).

Que le meilleur gagne !

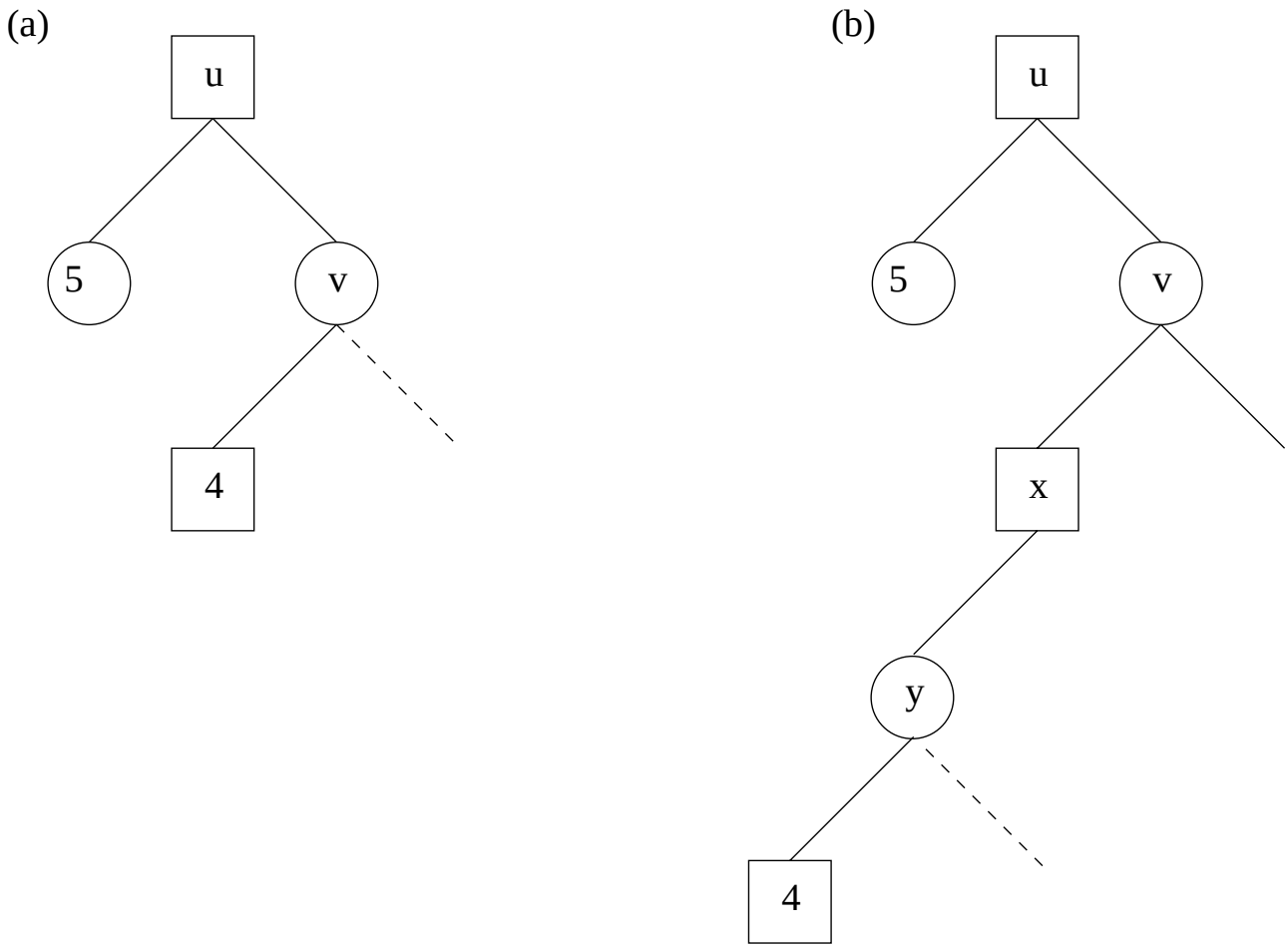


FIG. 10 – Coupure superficielle (a) et profonde (b)

5 Duel Tetris

5.1 Introduction

L'objet de ce projet est de présenter une extension au fameux jeu Tetris. cette extension est une intelligence artificielle qui joue au jeu en parallèle au joueur. Le but du jeu est :

- de comprendre les règles pour essayer de battre la machine,
- de construire une intelligence artificielle telle qu'il devienne extrêmement difficile de battre la machine,
- et enfin, en option, que la force de cette intelligence puisse être réglée (par exemple en utilisant un paramètre de 1 à 10).

Titre	**Année**	**Type de jeu**	**sous-type**
OXO	1952	IHM de jeu de Réflexion	Tic-tac-toe
PONG	1972	jeu de Sport	ping-pong
Space Invaders	1978	jeu de Tir	Shoot them up
Pac Man	1979	jeu de Labyrinthe	Plate-forme
Tetris	1984	jeu de Réflexion	Puzzle
Duel Tetris	2010	Jeu de Reflex	Puzzle

5.2 L'interface Homme-Machine

Dans cette partie, il s'agit de coder une interface pour jouer à Tetris. Un exemple peut être vu en figure 12.

5.2.1 Rappel des règles

- Le jeu contient 7 formes de bases (les tétrominos, c'est-à-dire les figures géométriques composées de quatre carrés) ayant chacune une couleur différente : *le carré (O), le bâton (I), le té (T), le èl (L), le èl inversé ou Gamma (J), le biais (Z) et le biais inversé (S)*.
- Ces formes descendent une par une dans un *puits* (*dix* cases de largeur et *vingt-deux* de haut).
- Le but du jeu étant de former des *lignes*.

Alphabeta (s : situation, α : entier, β : entier)
  m, t : entier ;
  Si s est une feuille
    Alors renvoyer $h(s)$;
  Si s est de type Max Alors
    $m \leftarrow \alpha$;
    Pout tout s' : situation $\in succ(s)$ Faire
      $t \leftarrow$ Alphabeta(s', m, β) ;
      Si $t > m$ Alors $m \leftarrow t$;
      Si $m \geq \beta$ Alors renvoyer m ;
    Fin Pour tout
    renvoyer m ;
  Si s est de type Min Alors
    $m \leftarrow \beta$;
    Pout tout s' : situation $\in succ(s)$ Faire
      $t \leftarrow$ Alphabeta(s', α, m) ;
      Si $t < m$ Alors $m \leftarrow t$;
      Si $m \leq \alpha$ Alors renvoyer m ;
    Fin Pour tout
    renvoyer m ;
Fin Alphabeta

FIG. 11 – Procédure Alphabeta

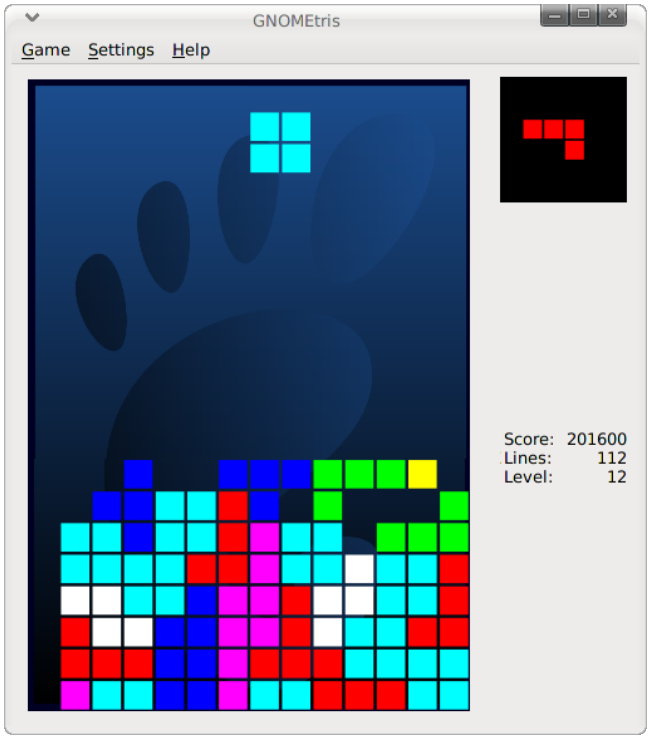


FIG. 12 – Une interface de Tetris.

– Toutes les *dix* lignes remplies, la vitesse de descente s'accélère d'un facteur $(1,1$ par exemple).

Le jeu se termine lorsqu'une pièce vient se poser sur une autre pièce qui a déjà atteint la hauteur maximum (c-à-d 22).

5.2.2 Codage de l'interface

Votre interface doit contenir une fenêtre graphique suffisamment grande pour contenir deux jeux de Tetris.

Chaque jeux de Tetris doit alors être composé d'une vue sur la pièce suivante (qui tombera dans le puits après celle déjà en chute (placée en haut par exemple).

Il doit être possible

– d'accélérer la vitesse de chute des pièces (d'un facteur deux par exemple)

– et de faire tourner les pièces d'un quart de tour.

5.3 L'Intelligence Artificielle

Dans cette partie, il s'agit de coder une interface pour jouer à Tetris.

5.3.1 Ajout de règles

Les règles énoncées précédemment ne présentent pas de notion de score.

En effet, comme le Tetris qui va être codé ici va être un *défi* entre l'homme et la machine, il n'y a pas besoin de score.

Par contre, nous allons ajouter la règle suivante, lorsque l'un des deux concurrents (homme ou machine) réussi à faire une ligne, celle-ci est bien retirée de son puits, mais elle va alors s'ajouter à la base du puits de l'adversaire mais ensuite cinq carrés seront retirés aléatoirement (si on envoie 2 lignes en même temps seulement quatre carrés seront retirés par lignes, pour 3 lignes, ce sera trois carrés par ligne et pour un envoie de 4 lignes, seulement deux carrés seront retirés aléatoirement par ligne).

Le jeu devient alors plus un jeu de reflex que de réflexion car le but devient d'envoyer le plus rapidement des lignes (même une seule) chez l'adversaire.

5.3.2 Mise en place de stratégies de décision

Vous êtes libres quant aux choix de stratégies mise en place par la machine pour former des lignes. Néanmoins, une modélisation objet est recommandé pour les tétrominos, chaque tétromino possède 4 représentations graphiques issues de ces transformation par la rotation de $\frac{\pi}{2}$, une méthode permet d'effectuer le déplacement du tétromino, une méthode permet sa rotation.

À chaque tétromino sont associées des configurations de lignes pour lesquelles son placement est optimum, et d'autre pour lequel son placement est plutôt bon. Vous devez classer ses positionnements possibles et choisir le placement qui a le meilleur score pour la configuration actuelle des lignes dans le puits (l'idéal étant que pour le placement, la ligne de surface ait un impact sur le score mais que soit également pris en compte la configuration des 2 ou 3 lignes sous la surface pour ne pas 'boucher' des places éventuellement disponibles pour une autre pièce, d'autant plus si c'est la pièce qui arrive juste après...).

6 Synthèse d'images par L-systèmes

Un L-System (ou système de Lindenmayer en bon français) est une grammaire formelle, permettant un procédé algorithmique, inventé en 1968 par le biologiste hongrois Aristid Lindenmayer qui consiste à modéliser le processus de développement et de prolifération de plantes ou de bactéries.

On pourra consulter l'entrée Wikipedia [1] pour une présentation succinte (page d'où est en fait tiré l'essentiel de cette présentation). La référence la plus complère est le livre ń*The Algorithmic beauty of plants*ż [2] de Przemyslaw Prusinkiewicz et Aristid Lindenmayer, 1990 où vous pourrez retrouver toutes ces informations et bien d'autres (les couleurs ne sont ici par exemple pas du tout traitées).

Plus précisément, un L-System est une grammaire formelle qui comprend :

– Un *alphabet* V. C'est l'ensemble des variables du L-System. V^* est l'ensemble des mots que l'on peut construire avec les symboles de V et V^+ l'ensemble des mots contenant au moins un symbole. (les notations V^* et V^+ sont standard en théorie des langages).
– Un ensemble de *constantes* S. Certains de ces symboles sont communs à tous les L-System (voir plus bas la Turtle interpretation).
– Un *axiome* ω choisi parmi V^+. Il s'agit de l'*état intial* choisi.
– Un ensemble de *règles* (on parle encore de *productions* en compilation), noté P, de réécriture de symboles de V.

Un L-System est alors noté (V, S, ω, P).

Illustrons ces définitions avec l'algue de Lindenmayer (un système simple permettant décrire le développement d'une algue) :

– Alphabet : $V = \{A, B\}$.
– Constantes : $S = \emptyset$.
– Axiome de départ : $\omega = A$.
– Règles : $\{(A \to AB), (B \to A)\}$.

La description informatique de ce système est donc :

[1] `http ://fr.wikipedia.org/wiki/L-System`
[2] `http ://algorithmicbotany.org/papers/abop/abop.pdf`

FIG. 13 – 3D L-System

```
Algue_de_Lindenmayer
{
Axiom A
A=AB
B=A
}
```

où `Algue_de_Lindenmayer` est le nom du L-System. En premier nous trouvons l'axiome ω, puis chaque règle de P (`A=AB` s'interprète comme tout symbole `A` devient un ńmotż `AB` à la génération suivante. Voici le résultat sur six générations :

- $n = 0$, `A`
- $n = 1$, `AB`
- $n = 2$, `ABA`
- $n = 3$, `ABAAB`
- $n = 4$, `ABAABABA`
- $n = 5$, `ABAABABAABAAB`
- $n = 6$, `ABAABABAABAABABAABABA`

Turtle interpretation

Cette chaîne de caractère est un mot sans signification en soi, mais qui peut fort bien se prêter à une interprétation graphique, en deux ou trois dimensions. Pour illustrer la manière de construire une plante avec un L-System, reprenons Wikipedia : *imaginons que nous avons un crayon à la main et qu'elle se balade sur la feuille sous nos ordres : "monte d'un cran, puis tourne de 20ř à gauche, déplace toi deux fois de un cran, m'morise ta position et avance encore d'un cran, lève-toi puis repose-toi sur la position mémorisée" et ainsi de suite... Il a donc fallu inventer des symboles variants $\in V$, ou constants $\in S$, pour permettre de guider cette main. Plusieurs d'entre eux ont été normalisés, ils font partie de ce qu'on appelle la "Turtle interpretation". Ce nom vient de la "tortue" du langage de programmation Logo qui fonctionne sur le même principe. En fait c'est cette tortue qui est votre main qui tient le crayon.*

Voici les signes couramment utilisés :

- '`F`' : Se déplacer d'un pas unitaire ($\in V$).
- '+' : Tourner à gauche d'angle α ($\in S$).
- '−' : Tourner à droite d'un angle α ($\in S$).
- '`&`' : Pivoter vers le bas d'un angle α ($\in S$).
- '`^`' : Pivoter vers le haut d'un angle α ($\in S$).
- '<' : Roulez vers la gauche d'un angle α ($\in S$).
- '>' : Roulez vers la droite d'un angle α ($\in S$).
- '`|`' : Tourner sur soi-même de $180°$ ($\in S$).
- '`[`' : Sauvegarder la position courante ainsi que le contexte graphique($\in S$)
- '`]`' : Restaurer la dernière position sauvée ainsi que le contexte graphique ($\in S$).

Illustrons cela avec la courbe de Koch :

- Alphabet : $V = \{F\}$.

– Constantes : $S = \{+, -\}$.
– Axiome de départ : $\omega = F$.
– Règles : $\{(F \rightarrow F + F - F - F + F)\}$.

Une description informatique de ce système est donc :

```
La_Courbe_de_Koch
{
angle 90
Axiom F
F=F+F-F-F+F
}
```

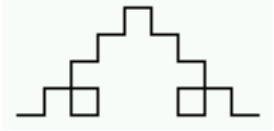
où `angle` détermine que l'on tourne de $90°$ avec les symboles `+` et `-`.

Voici le résultat sur deux générations :

– $n = 0$, `F`

– $n = 1$, `+F-F-F+F`

– $n = 2$, `F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F`



SOL-System

Le système qui nous avons évoqué est déterministe, *i.e.*, il n'offre qu'une seule évolution possible depuis l'axiome à la génération n. Un SOL-System fait appel aux probabilités, il est aussi appelé système non-déterministe. Il est alors possible de déterminer plusieurs transformations pour un symbole. Chaque possibilité sera pondérée pour pouvoir donner priorité à certaines transformations par rapport à d'autres.

Par exemple, pour une plante stochastique :

```
Une_Plante_Stochastique
{
angle 20
axiom X
X=(0.2)F[++X]F[-X]+X
X=(0.8)F[+X]F[-X]+X
F=(1.0)FF
}
```

où `(0.2)`, `(0.8)` et `(1.0)` représentent les poids de chaque transformation possible de `X` et de `F`.

Voici un résultat possible sur deux générations :

– $n = 0$, `X`
– $n = 1$, `F[+X]F[-X]+X`
– $n = 2$, `F[+F[++X]F[-X]+X]F[-F[++X]F[-X]+X]+F[++X]F[-X]+X`

COntext-sensitive

Les systèmes précédents ne peuvent pas simuler l'interaction de parties d'une plante car ils sont context-free, 'emphi.e., chaque partie se développe indépendamment des autres parties. Un L-System context-sensitive résout ce problème en prenant en compte ce qui précéde ou succède à une partie, c'est-à-dire un symbole. Un tel système est appelé (k, l)-System, le contexte de gauche est un "mot" de longueur k et celui de droite un "mot" de longueur l. Pour expliquer la manière dont se lisent les regles voici deux exemples :

Un signal *acropète* :
– Variable : $V = \{A, B\}$.
– Constantes : $S = \{+, -, [,], <\}$.
– Axiome : $\omega = B[+A]A[-A]A[+A]A$
– Règles : $\{(B < A \rightarrow B)\}$.

La règle se comprend ainsi : si un symbole `A` est **précédé** d'un symbole `B`, alors ce `A` devient un `B` à la génération suivante.

Un signal *basipète* :
- Variable : $V = \{A, B\}$.
- Constantes : $S = \{+, -, [,], <\}$.
- Axiome : $\omega = A[+A]A[-A]A[+A]B$
- Règles : $\{(B > A \rightarrow B\})$.

La règle se comprend ainsi : si un symbole `A` est **suivi** d'un symbole `B`, alors ce `A` devient un `B` à la génération suivante.

Extension à la 3D

Cette ńturtle interpretationż peut être exploitée en trois dimensions grâce aux idées de Harold Abeson et Andrea diSessa dans leur ouvrage commun, ń Turtle geometry : the computer as a medium for exploring mathematic ż. L'orientation est représentée par trois vecteurs $\vec{H}$, $\vec{L}$ et $\vec{U}$ avec $\vec{H} \times \vec{L} = \vec{U}$.
- $\vec{H}$ pour *turtle heading*. Il s'agit du regard de la tortue.
- $\vec{U}$ pour *up*. Il s'agit de la direction vers laquelle se dirige la tortue.
- $\vec{L}$ pour *left*. Il s'agit de la gauche de cette tortue.

La rotation de la tortue se note alors $[\vec{H}'\vec{L}'\vec{U}'] = [\vec{H}\vec{L}\vec{U}]R$, où R est une matrice 3×3. Les rotations d'un angle α autour des axes $\vec{U}$, $\vec{L}$ ou $\vec{H}$ sont représentées par les matrices :

$$RU(\alpha) = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$RL(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix}$$

$$RH(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

Les symboles prennent maintenant la signification suivante :
- `'+'` : Tourner à gauche d'un angle α, en utilisant la matrice de rotation $RU(\alpha)$.
- `'-'` : Tourner à droite d'un angle α, en utilisant la matrice de rotation $RU(-\alpha)$.
- `'&'` : Pivoter vers le bas d'un angle α, en utilisant la matrice de rotation $RL(\alpha)$.
- `'^'` : Pivoter vers le haut d'un angle α, en utilisant la matrice de rotation $RL(-\alpha)$.
- `'\'` : Rouler à droite d'un angle α, en utilisant la matrice de rotation $RH(\alpha)$.
- `'/'` : Rouler à gauche d'un angle α, en utilisant la matrice de rotation $RH(-\alpha)$.
- `'|'` : Tourner autour de lui-même de $180°$, en utilisant la matrice de rotation $RU(180°)$.

Travail demandé

Le travail minimum demandé est :
- Implémentation et rendu graphique des systèmes de Lindenmayer déterministes.
- Implémentation et rendu graphique des systèmes de Lindenmayer stochastiques.
- Implémentation et rendu graphique des systèmes de Lindenmayer context-sensitive.

Noter qu'il vous sera nécessaire de développer un module pour lire des descriptions informatiques de Lindenmayer.

Pour les plus avancés d'entre vous.
- Gestion des couleurs.
- Gestion de la 3D et rendu graphique 2D.
- Sauvegarde d'images.

7 Logiciel de traitement d'images

L'objectif est de créer un mini-logiciel de retouche d'images en s'inspirant de Photoshop.

7.1 Interface et utilisation du logiciel

En raison du temps limité, le logiciel de traitement d'images sera implémenté de manière *simple*. On abandonnera donc
- la création de menus et de boîtes de dialogue,
- la gestion des calques

pour se concentrer sur l'essentiel.

Cependant on veillera à une interface graphique suffisamment conviviale et intuitive.

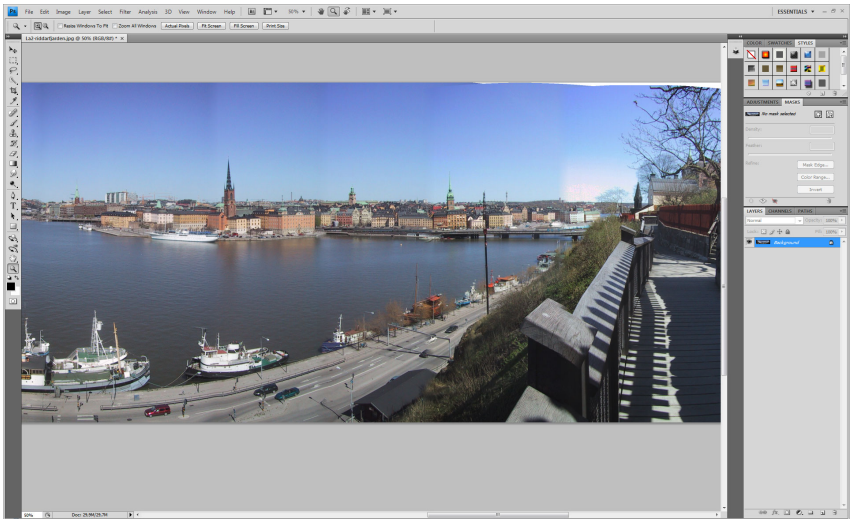


FIG. 14 – Photoshop avec une image panoramique.

7.1.1 Chargement de l'image

Une alternative plutôt bonne est de faire un drag-and-drop de l'image vers le programme pour l'affichage. Pour cela, on implémentera la fonction `main` de la manière suivante

```
#include <iostream>
using namespace std;

int main(int argc, char **argv)
{
  if(argc != 2)
  {
    cerr << "Erreur d'utilisation !" << endl;
    cerr << "Usage en mode commande: Photoshop [nom du fichier image]" << endl;
    cerr << "Usage en mode standard: Drag-and-drop du fichier image sur le programme" << endl;
    return -1; // Erreur d'utilisation
  }
  ChargerImage(argv[1]); // argv[1] est une chaîne de caractères contenant le nom de l'image à
      charger.
  // etc
  return 0;
}
```

7.1.2 Interface graphique

On utilisera une seule fenêtre principale. Comme dans *Paint* dans *Windows*, on pourra par exemple
- réserver la partie centrale de la fenêtre pour afficher l'image
- délimiter une bande gauche sur laquelle on dessinera des icônes décrivant les principaux outils de retouche
- réserver une bande en haut/bas de la fenêtre pour afficher une palette de couleurs et autre

7.2 Traitement d'images

7.2.1 Les fonctionnalités minimales

- Implémenter une classe image couleur raisonnable.
- Sauvegarder une image.
- Ne pas afficher tout tout le temps.
- Deux types de fonctionnalités :
  - Des algorithmes appliqués automatiquement à toute l'image comme le négatif ou le flou.
  - Une retouche interactive de type pinceau.

7.2.2 Les techniques classiques pour le traitement d'images

Voici ce qu'il serait souhaitable de faire ensuite
- Zoom d'image +/-
- Changement de contraste

- Pipette pour récupérer la couleur d'un pixel de l'image
- Transformation d'image (rotation, translation, homographie)
- Une operation de sélection rectangulaire l'image pour modifier une région précise de l'image.
- D'autres algorithmes plus spécifiques (netteté, déformations, contours, débruitage, etc.) : demandez à vos enseignants spécialistes d'images.
- Doigt pour flouter/corriger localement une zone de l'image à la manière du pinceau.
- Choisir des brosses différentes pour le pinceau.
- D'autres types de retouche : clone par exemple (utile pour truquer des images)

8 Autre

Vous pouvez proposer des sujets (à faire valider par le responsable du cours !) mais ne le faîtes que si vous vous sentez à l'aise en programmation et si vous êtes autonome.