Algorithmique & Programmation
Semestre 2 ST
Les matrices de variables et applications pratiques

Problématique

- Emploi de "matrices" de variables dans le cadre de nombreux problèmes
- Matrice: Tableau à deux indices

Déclaration de tableaux à deux indices en langage algorithmique et en langage Java

- Chapitre "Tableaux de variables" pour la syntaxe en langage algorithmique ainsi que pour la syntaxe en langage Java.

Matrices pour les Mathématiques

- Matrice: Tableau de variables à deux indices
  - Composantes du même type
  - Premier indice: Numéro de ligne
  - Second indice: Numéro de colonne

× × × × ×
× × × × ×
× × × × ×

nombre lignes = 3, nombre colonnes = 5

- Vecteur: Tableau de variables à deux indices dont l'une des 2 tailles est égale à 1
  - Matrice n*1
  - Matrice 1*n

×

× × × × ×

x nombre lignes = 1, nombre colonnes = 5

x

nombre lignes = 3, nombre colonnes = 1

- Matrices et vecteurs: "Objets" mathématiques de même catégorie: Tableau à 2 dimensions
  - Si tailles compatibles, objets associables au moyen d'opérations matricielles:
    - Addition
    - Soustraction
    - Multiplication (composition)

- En programmation informatique, vecteur usuellement géré de manière simplifiée en utilisant un tableau à une dimension
  -> Pas de second indice toujours égal à 0 à "traîner"

- Matrice carrée: Matrice avec des nombres de lignes et de colonnes identiques

- En mathématiques utilisation fréquente des notions de matrice et de vecteur:
  - Ensemble de valeurs: Un tableau unidimensionnel de taille arbitraire
  - Position dans un espace: Un tableau unidimentionnel de taille égale à la dimension de l'espace
  - Transformation géométrique: Un tableau bidimentionnel carré de taille égale à la dimension de l'espace
  - Equation linéaire: Un tableau unidimentionnel de taille égale au nombre de variables de l'équation linéaire + 1
  - Système d'équations linéaires: Tableau bidimentionnel de taille ne*nv + tableau unidimentionnel de taille ne
    où ne = nombre d'équations et nv = nombre d'inconnues
  - ...

- Utilisation fréquente de matrices carrées

Exemples

- **Mathématiques matricielles: Produit matrice par vecteur**

- On considère un vecteur $\vec{V}$ de taille n et une matrice carrée M de taille n*n.
  Le vecteur $\vec{W}$ produit de M par $\vec{V}$ (noté $\vec{W} = M.\vec{V}$) est calculé selon la formule:
  $w_i = \sum_{k=1}^{n} m_{ik} v_k$ pour i de 1 à n
  où les w_i sont les composantes du vecteur $\vec{W}$, les m_{ik} sont les composantes de la matrice M et les v_k sont les composantes du vecteur $\vec{V}$.

- Présentation intuitive du calcul du produit matrice-vecteur: La composante i du vecteur résultat est le produit de la $i^{ème}$ ligne de la matrice par le vecteur $\vec{V}$.

  Exemple:

$$\begin{pmatrix} 2 & 3 & 5 & 1 \\ -1 & 4 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 4 \\ -2 \\ 0{,}5 \end{pmatrix} = \begin{pmatrix} 2{,}5 \\ 10 \\ 0 \\ 8 \end{pmatrix}$$

```
constante N : entier <- ...

{ Methode de calcul du produit d'un tableau   }
```

```
{ carre de taille N.N par un tableau          }
{ a 1 indice de taille N                      }
{ Resultat: Un tableau a 1 indice de taille N }

Tableau [N] de reel fonction produitMatriceVecteur(m,v)
   Données
    m : Tableau [N][N] de reel
    v : Tableau [N] de reel
   Locales
    i : entier
    j : entier
    w : Tableau [N] de reel
   pour i de 0 à N-1 faire
    w[i] <- 0.0
     pour j de 0 à N-1 faire
      w[i] <- w[i] + m[i][j]*v[j]
     fait
   fait
   retourner w
fin fonction
```

```
/* Methode de calcul du produit d'un tableau   */
/* carre de taille n.n par un tableau          */
/* a 1 indice de taille n                      */
/* Resultat: Un tableau a 1 indice de taille n */

static double [] produitMatriceVecteur(double [][] m,
                                       double [] v) {
   int i;
   int j;
   int n;
  n = v.length;
   double [] w = new double[n];
   for ( i = 0 ; i < n ; i = i+1 ) {
    w[i] = 0.0;
     for ( j = 0 ; j < n ; j = j+1 ) {
      w[i] = w[i] + m[i][j]*v[j]; } }
   return w;
}
```

ProduitMatriceVecteur.lda
ProduitMatriceVecteur.java
Exemple d'exécution

- **Mathématiques matricielles: Produit matrice par matrice**

- On considère une matrice M1 de taille n*m et une matrice M2 de taille m*p.
  La matrice M produit de M1 par M2 (noté M = M1.M2) est une matrice de taille n*p et est calculée au moyen de la formule:
  $m_{ij} = \sum_{k=1}^{m} m1_{ik} m2_{kj}$ pour i de 1 à n et j de 1 à p
  où les m_{ij} sont les composantes de la matrice M, les $m1_{ik}$ sont les composantes de la matrice M1 et les $m2_{kj}$ sont les composantes de la matrice M2.

- Présentation intuitive du calcul du produit matrice-matrice: La composante ij de la matrice résultat est le produit de la $i^{ème}$ ligne de la première matrice par la $j^{ème}$ colonne de la seconde matrice.

  Exemple:

$$\begin{pmatrix} 1 & -1 & 2 & -3 \\ 0 & 0 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 5 & 0 & 1 \\ 4 & 0 & 0 \\ 3 & 1 & 2 \\ -2 & 3 & -1 \end{pmatrix} = \begin{pmatrix} 13 & -7 & 8 \\ -1 & 7 & 0 \end{pmatrix}$$

```
constante N : entier <- ...
constante M : entier <- ...
constante P : entier <- ...

{ Methode de calcul du produit d'un tableau   }
{ de dimension 2 de taille N.M                }
{ par un tableau de dimension 2 de taille M.P }
{ Resultat: Un tableau de dimension 2         }
{ de taille N.P                               }

Tableau [N][P] de reel fonction produitMatriceMatrice(m1,m2)
   Données
    m1 : Tableau [N][M] de reel
    m2 : Tableau [M][P] de reel
   Locales
    i : entier
    j : entier
    k : entier
    r : Tableau [N][P] de reel
   pour i de 0 à N-1 faire
     pour j de 0 à P-1 faire
      r[i][j] <- 0.0
       pour k de 0 à M-1 faire
        r[i][j] <- r[i][j] + m1[i][k]*m2[k][j]
       fait
     fait
   fait
   retourner r
fin fonction
```

```
/* Methode de calcul du produit d'un tableau   */
/* de dimension 2 de taille n.m                */
/* par un tableau de dimension 2 de taille m.p */
/* Resultat: Un tableau de dimension 2         */
/* de taille n.p                               */

static double [][] produitMatriceMatrice(double [][] m1,
                                         double [][] m2) {
   int i;
   int j;
   int k;
   int n;
   int m;
   int p;
  n = m1.length;
  m = m2.length;
  p = m2[0].length;
   double [][] r = new double[n][p];
   for ( i = 0 ; i < n ; i = i+1 ) {
     for ( j = 0 ; j < p ; j = j+1 ) {
      r[i][j] = 0.0;
       for ( k = 0 ; k < m ; k = k+1 ) {
        r[i][j] = r[i][j] + m1[i][k]*m2[k][j]; } } }
   return r;
}
```

ProduitMatriceMatrice.lda
ProduitMatriceMatrice.java
Exemple d'exécution

- **Algèbre linéaire: Résolution d'un système de n équations linéaires à n inconnues**

- On considère le système de n équations linéaires à n inconnues:

$$\begin{cases} a_{11} * v_1 + a_{12} * v_2 + \dots + a_{1n} * v_n = b_1 \\ a_{21} * v_1 + a_{22} * v_2 + \dots + a_{2n} * v_n = b_2 \\ \\ a_{n1} * v_1 + a_{n2} * v_2 + \dots + a_{nn} * v_n = b_n \end{cases}$$

où les inconnues sont les v_i et les constantes sont les a_{ij} et les b_i.

- Présentation possible sous la forme matricielle A.V = B:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

où A est une matrice carrée de taille n*n, V et B sont des vecteurs de taille n.

- Résolution de ce système:
  - Détermination, s'il existe, du seul vecteur des v_i qui vérifie l'ensemble de ces équations et donc du vecteur V qui vérifie le produit matriciel.
  - Exemples d'application pratique
    - En deux dimensions: Trouver la position de l'intersection de deux droites
    - En trois dimensions: Trouver la position de l'intersection entre une droite et un plan
    - En dimension n avec n très grand (> 1000000): En informatique graphique, calculer l'éclairage d'une scène par radiosité

- Méthode de résolution par pivot de Gauss

  - Une première étape de transformation par "triangularisation":
  - Transformation de la matrice A par traitement (substitution) de ses lignes de manière à remplir sa 1/2 matrice triangulaire inférieure stricte avec des 0.0
  - Transformation concourante du vecteur B pour que le système d'équations A.V = B reste équivalent
    - n-1 itérations numérotées i consistant à soustraire à chaque ligne j à partir de la ligne i+1 le produit de la ligne i par le facteur fact tel que fact*m[i][i] = m[i][j] -> fact = m[i][j]/m[i][i]
    - Calcul de fact impossible si, lors d'une itération, m[i][i] = 0.0 (division par zéro)
      -> Permutation la ligne i avec la première ligne k où m[k][i] est différent de 0.0
      -> Si tous les m[k][i] égaux à 0.0 alors pas de solution unique (aucune solution ou une infinité de solutions)

  Exemple:

Les matrices A et B sont $\begin{bmatrix} 1.0 & 0.0 & 1.0 & -1.0 & 1.0 \\ 4.0 & 0.0 & 5.0 & 1.0 & -1.0 \\ 3.0 & 1.0 & 2.0 & 0.0 & 3.0 \\ -2.0 & 3.0 & -2.0 & 16.0 & 3.0 \\ 6.0 & 5.0 & 0.0 & -3.0 & 19.0 \end{bmatrix}$ et $\begin{bmatrix} 1.0 \\ 2.0 \\ -1.0 \\ 4.0 \\ 3.0 \end{bmatrix}$.

1. Etape n°1

I. Traitement de la ligne 2 par soustraction de 4.0 fois la ligne 1:

$$\begin{bmatrix} 1.0 & 0.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 5.0 & -5.0 \\ 3.0 & 1.0 & 2.0 & 0.0 & 3.0 \\ -2.0 & 3.0 & -2.0 & 16.0 & 3.0 \\ 6.0 & 5.0 & 0.0 & -3.0 & 19.0 \end{bmatrix} \text{ et } \begin{bmatrix} 1.0 \\ -2.0 \\ -1.0 \\ 4.0 \\ 3.0 \end{bmatrix}.$$

II. Traitement de la ligne 3 par soustraction de 3.0 fois la ligne 1:

$$\begin{bmatrix} 1.0 & 0.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 5.0 & -5.0 \\ 0.0 & 1.0 & -1.0 & 3.0 & 0.0 \\ -2.0 & 3.0 & -2.0 & 16.0 & 3.0 \\ 6.0 & 5.0 & 0.0 & -3.0 & 19.0 \end{bmatrix} \text{ et } \begin{bmatrix} 1.0 \\ -2.0 \\ -4.0 \\ 4.0 \\ 3.0 \end{bmatrix}.$$

III. Traitement de la ligne 4 par soustraction de -2.0 fois la ligne 1:

$$\begin{bmatrix} 1.0 & 0.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 5.0 & -5.0 \\ 0.0 & 1.0 & -1.0 & 3.0 & 0.0 \\ 0.0 & 3.0 & 0.0 & 14.0 & 5.0 \\ 6.0 & 5.0 & 0.0 & -3.0 & 19.0 \end{bmatrix} \text{ et } \begin{bmatrix} 1.0 \\ -2.0 \\ -4.0 \\ 6.0 \\ 3.0 \end{bmatrix}.$$

IV. Traitement de la ligne 5 par soustraction de 6.0 fois la ligne 1:

$$\begin{bmatrix} 1.0 & 0.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 5.0 & -5.0 \\ 0.0 & 1.0 & -1.0 & 3.0 & 0.0 \\ 0.0 & 3.0 & 0.0 & 14.0 & 5.0 \\ 0.0 & 5.0 & -6.0 & 3.0 & 13.0 \end{bmatrix} \text{ et } \begin{bmatrix} 1.0 \\ -2.0 \\ -4.0 \\ 6.0 \\ -3.0 \end{bmatrix}.$$

2. Etape n°2

I. Permutation des lignes 2 et 3 car la composante d'indice (2,2) est égale à 0.0:

$$\begin{bmatrix} 1.0 & 0.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 1.0 & -1.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 5.0 & -5.0 \\ 0.0 & 3.0 & 0.0 & 14.0 & 5.0 \\ 0.0 & 5.0 & -6.0 & 3.0 & 13.0 \end{bmatrix} \text{ et } \begin{bmatrix} 1.0 \\ -4.0 \\ -2.0 \\ 6.0 \\ -3.0 \end{bmatrix}.$$

II. Traitement des lignes 4 et 5:

$$\begin{bmatrix} 1.0 & 0.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 1.0 & -1.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 5.0 & -5.0 \\ 0.0 & 0.0 & 3.0 & 5.0 & 5.0 \\ 0.0 & 0.0 & -1.0 & -12.0 & 13.0 \end{bmatrix} \text{ et } \begin{bmatrix} 1.0 \\ -4.0 \\ -2.0 \\ 18.0 \\ 17.0 \end{bmatrix}.$$

3. Etape n°3

I. Traitement des lignes 4 et 5:

$$\begin{bmatrix} 1.0 & 0.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 1.0 & -1.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 5.0 & -5.0 \\ 0.0 & 0.0 & 0.0 & -10.0 & 20.0 \\ 0.0 & 0.0 & 0.0 & -7.0 & 8.0 \end{bmatrix} \text{ et } \begin{bmatrix} 1.0 \\ -4.0 \\ -2.0 \\ 24.0 \\ 15.0 \end{bmatrix}.$$

4. Etape n°4

I. Traitement de la ligne 5:

$$\begin{bmatrix} 1.0 & 0.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 1.0 & -1.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 5.0 & -5.0 \\ 0.0 & 0.0 & 0.0 & -10.0 & 20.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -6.0 \end{bmatrix} \text{ et } \begin{bmatrix} 1.0 \\ -4.0 \\ -2.0 \\ 24.0 \\ -1.8 \end{bmatrix}.$$

- Une seconde (et dernière) étape: Calcul de V
  - v_n d'ores et déjà calculable car sur la ligne n de la matrice A, il ne reste que des 0.0 sauf pour la dernière valeur d'indice (n,n)
    -> $v_n = b_n/a_{nn}$
  - Calcul de v_{n-1} possible car, v_n étant connu, toutes les valeurs nécessaires à son calcul sont maintenant connues.
  - Calcul de v_{n-2} possible car, v_n et v_{n-1} étant connus, toutes les valeurs nécessaires à son calcul sont maintenant connues.
  - Poursuite du calcul des v du calcul des v_i jusqu'à v_1.

Sur l'exemple précédent:

- v_5 = -1.8/-6.0 = 0.3
- v_4 = (24.0 - 20.0*0.3)/-10.0 = -1.8
- v_3 = (-2.0 + 5.0*0.3 + 5.0*1.8)/1.0 = 8.5
- v_2 = (-4.0 + 0.0*0.3 + 3.0*1.8 + 1.0*8.5)/1.0 = 9.9
- v_1 = (1.0 - 1.0*0.3 - 1.0*1.8 - 1.0*8.5 + 0.0*9.9)/1.0 = -9.6

```
constante N : entier <- ...

{ Resolution d'une systeme de N equations     }
{ lineaires a N inconnues                     }
{ par la methode de du pivot Gauss            }

{ Permutation de Gauss                        }

action permutation(l,m,v)
   Données
    l : entier
   Données / Résultats
    m : Tableau [N][N] de réel
    v : Tableau [N] de réel
   Locales
    ll : entier
    i : entier
    aux : reel
  ll <- l
   tant que m[ll][l] = 0.0 faire
    ll <- ll+1
   fait
   pour i de l à N-1 faire
```

```
    aux <- m[l][i]
    m[l][i] <- m[ll][i]
    m[ll][i] <- aux
   fait
  aux <- v[l]
  v[l] <- v[ll]
  v[ll] <- aux
fin action

{ Transformation de Gauss                          }

action transformation(m,v)
   Données / Résultats
    m : Tableau [N][N] de réel
    v : Tableau [N] de réel
   Locales
    i : entier
    j : entier
    k : entier
    facteur : reel
   pour i de 1 à N-1 faire
     si m[i-1][i-1] = 0.0 alors
      permutation(i-1,m,v)
     fsi
     pour j de i à N-1 faire
      facteur <- m[j][i-1]/m[i-1][i-1]
       pour k de i-1 à N-1 faire
        m[j][k] <- m[j][k] - m[i-1][k]*facteur
       fait
      v[j] <- v[j] - v[i-1]*facteur
     fait
   fait
fin action

{ Extraction de Gauss                              }

Tableau [N] de réel fonction extraction(a,b)
   Données
    a : Tableau [N][N] de réel
    b : Tableau [N] de réel
   Locales
    i : entier
    j : entier
    v : Tableau [N] de reel
  v[N-1] <- b[N-1]/a[N-1][N-1]
   pour i de N-2 à 0 pas -1 faire
    v[i] <- b[i]
     pour j de N-1 à j+1 pas -1 faire
      v[i] <- v[i] - v[j]*a[i][j]
     fait
    v[i] <- v[i]/a[i][i]
   fait
   retourner v
fin fonction

{ Resolution de Gauss                              }

Tableau [N] de réel fonction resolution(a,b)
   Données
    a : Tableau [N][N] de réel
    b : Tableau [N] de réel
   Locales
    v : Tableau [N] de reel
  transformation(a,b)
  v <- extraction(a,b)
```

```
   retourner v
fin fonction
```

```
/* Methode de clonage d'un tableau              */
/* a 1 indice de double                         */

static double [] clone(double [] t) {
   int i;
   int n = t.length;
   double [] nt = new double[n];
   for ( i = 0 ; i < n ; i = i+1 ) {
    nt[i] = t[i]; }
   return nt;
}

/* Methode de clonage d'un tableau              */
/* a 2 indices de double                        */

static double [][] clone(double [][] t) {
   int i;
   int j;
   int n = t.length;
   int m = t[0].length;
   double [][] nt = new double[n][m];
   for ( i = 0 ; i < n ; i = i+1 ) {
     for ( j = 0 ; j < m ; j = j+1 ) {
      nt[i][j] = t[i][j]; } }
   return nt;
}

/* Permutation de Gauss                         */

static void permutation(int l,double [][] m,double [] v) {
   int ll;
   int i;
   int n = v.length;
   double aux;
  ll = l;
   while ( m[ll][l] == 0.0 ) {
    ll = ll+1; }
   for ( i = l ; i < n ; i = i+1 ) {
    aux = m[l][i];
    m[l][i] = m[ll][i];
    m[ll][i] = aux; }
  aux = v[l];
  v[l] = v[ll];
  v[ll] = aux;
}

/* Transformation de Gauss                      */

static void transformation(double [][] m,
                           double [] v) {
int n = v.length;
int i;
int j;
int k;
double facteur;
for ( i = 1 ; i < n ; i++ ) {
   if ( m[i-1][i-1] == 0.0 )
    permutation(i-1,m,v);
   for ( j = i ; j < n ; j++ ) {
    facteur = m[j][i-1]/m[i-1][i-1];
     for ( k = i-1 ; k < n ; k++ ) {
      m[j][k] = m[j][k] - m[i-1][k]*facteur; }
    v[j] = v[j] - v[i-1]*facteur;  } }
```

```
}

/* Extraction de Gauss                          */

static double [] extraction(double [][] a,
                            double [] b) {
   int i;
   int j;
   int n = b.length;
   double [] v = new double[n];
  v[n-1] = b[n-1]/a[n-1][n-1];
   for ( i = n-2 ; i >= 0 ; i = i-1 ) {
    v[i] = b[i];
     for ( j = n-1 ; j > i ; j = j-1 ) {
      v[i] = v[i] - v[j]*a[i][j]; }
    v[i] = v[i]/a[i][i]; }
   return v;
}

/* Resolution de Gauss                          */

static double [] resolution(double [][] a,
                            double [] b) {
   double [] v;
  transformation(a,b);
  v = extraction(a,b);
   return v;
}

/* Resolution de Gauss                          */
/* sur une copie des 2 tableaux en parametres   */

static double [] resolutionGauss(double [][] a,
                                 double [] b) {
   double [][] ca = clone(a);
   double [] cb = clone(b);
   double [] v;
  v = resolution(ca,cb);
   return v;
}
```

PivotGauss.lda
PivotGauss.java
Exemple d'exécution

Matrices pour la Physique

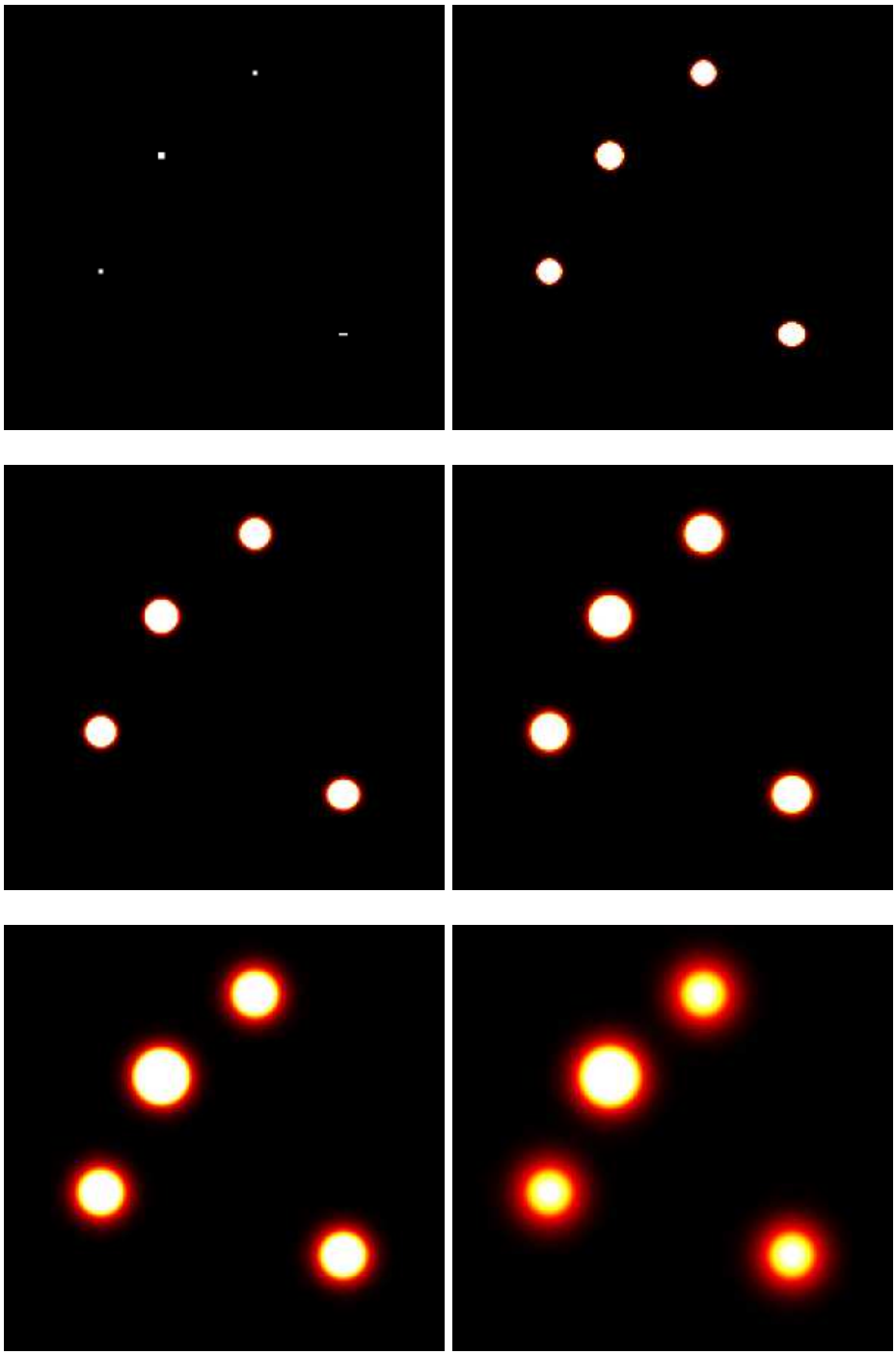
- Large appel à l'informatique comme outil de simulation
- Pourquoi simuler?
  - Moins cher que les tests réels (crashs automobiles, analyses d'écoulements turbulents, ...)
  - Réalisation de prévisions (météo, évolution climatique, collision de galaxies, ...)
  - Analyse de phénomènes qu'il n'est pas possible de tester dans la réalité (formation des étoiles, super-novae, essais nucléaires, ...)
  - ...
- Phénomènes à simuler fréquement décrits par des équations continues pour lesquelles il n'existe pas de méthode de résolution analytique (i.e. pas de solution présentable sous la forme d'une formule simple, voire complexe)
- Méthodes numériques de résolution de certaines classes de problèmes permettant de trouver une solution à partir d'une discrétisation du champ de travail
  - Champ unidimentionnel: Appel à des tableaux

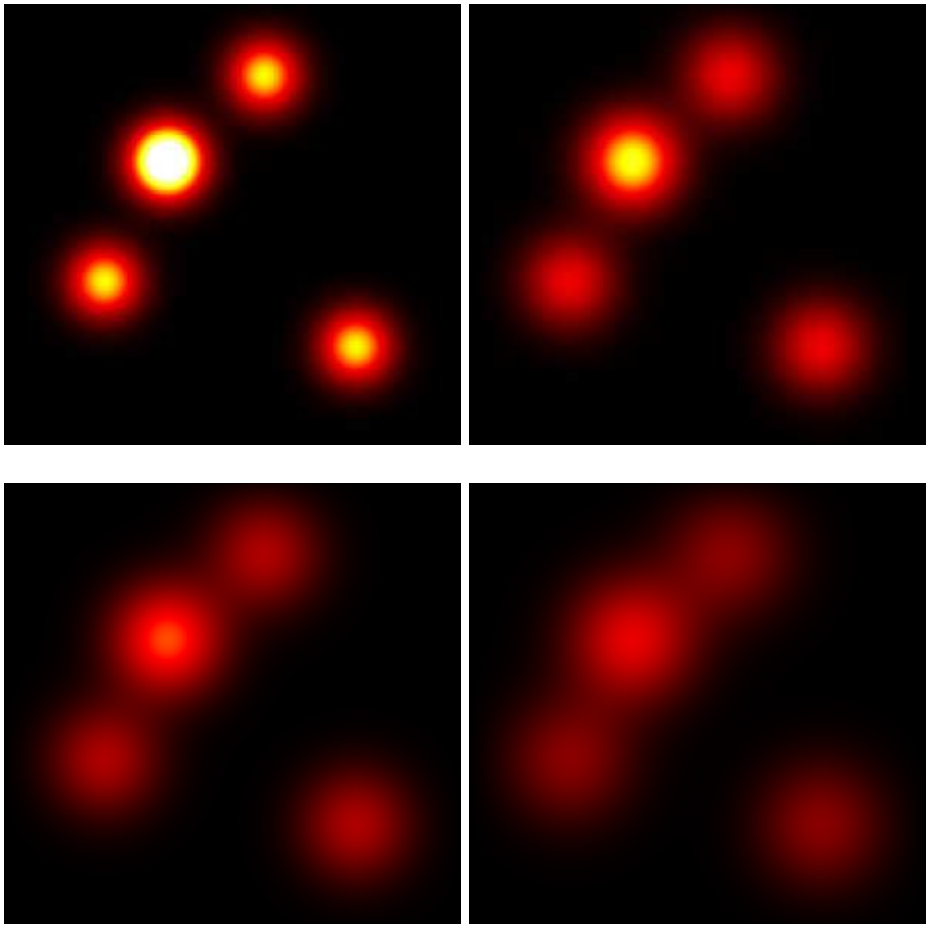
- Champ bidimentionnel: Utilisation de matrices
- Dimension 3: Tableaux à 3 indices

Exemples

- Propagation de la chaleur au sein d'une barre métallique
  - Problème en dimension 1
  - Décomposition (discrétisation) de la barre en une suite de morceaux pour chacun desquels la température est considérée uniforme
  - Valeurs stockées et gérées dans un tableau à 1 indice
  - Augmentation du pas de discrétisation -> Obtention de résultats plus précis au prix d'un temps de calcul plus long

- Propagation de la chaleur au sein d'une plaque métallique
  - Problème en 2 dimensions
  - Décomposition de la plaque en une grille de cellules pour chacune desquelles la température est considérée uniforme
  - Valeurs stockées dans un tableau à 2 indices

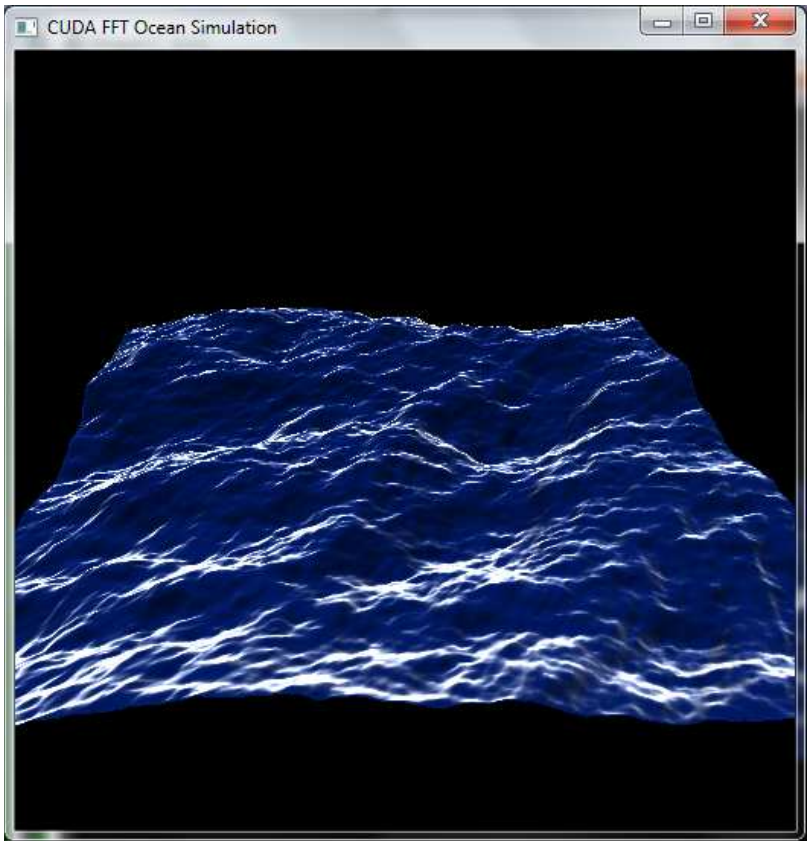


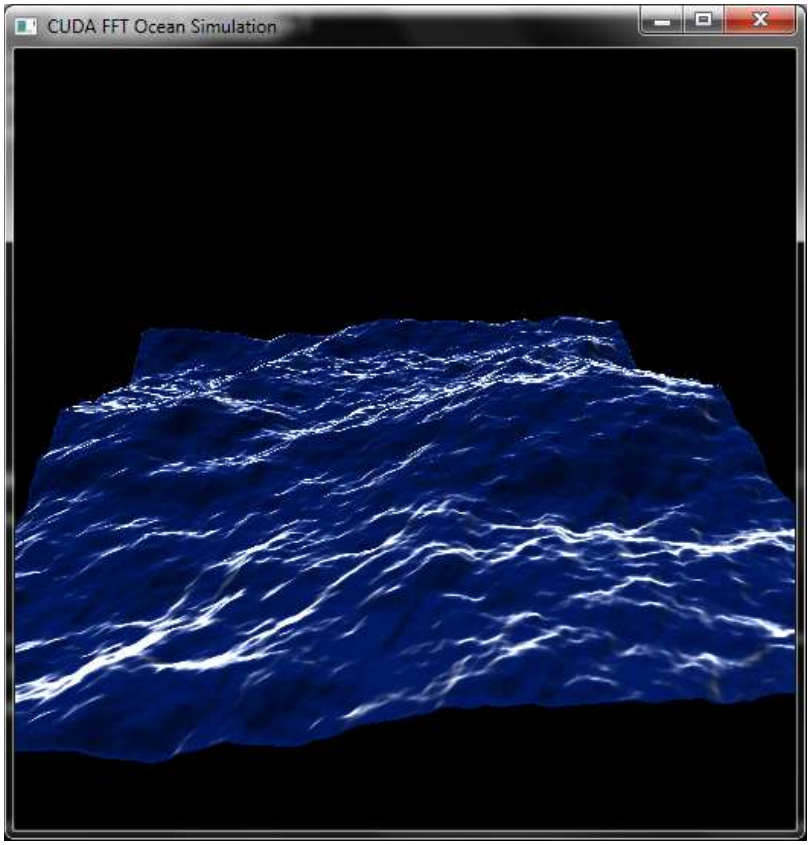


Simulation de l'évolution de la température d'une plaque métallique sur laquelle sont créés des points chauds

Exemple d'exécution

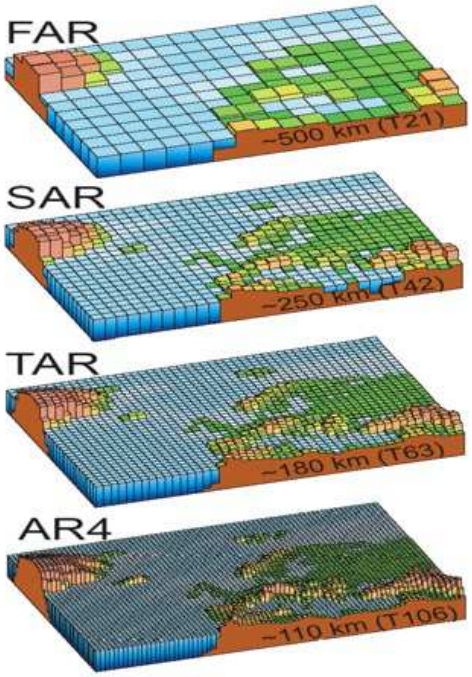
- Simulation de la surface océanique
  - Problème en 2 dimensions



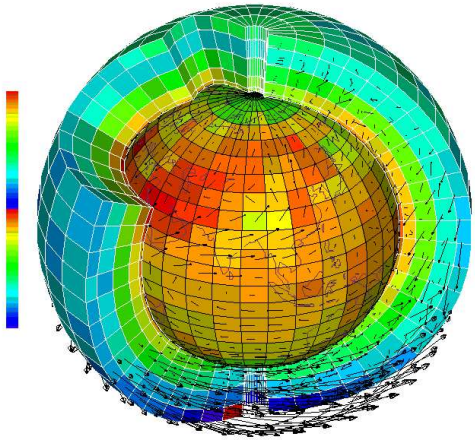


Simulation de surface océanique
Kit de développement CUDA (NVidia)

- Evolution climatique
  - Premiers modèles: Modèles en deux dimensions
  - Surface de la Terre décomposée en longitude et en latitude en un maillage "carré"
  - Informations relatives aux cellules de ce maillage stockées dans une matrice de variables
  - Etude de l'évolution du climat en considérant chaque cellule comme une entité atomique de caractéristiques uniformes
  - Modèles plus complexes: Trois dimensions

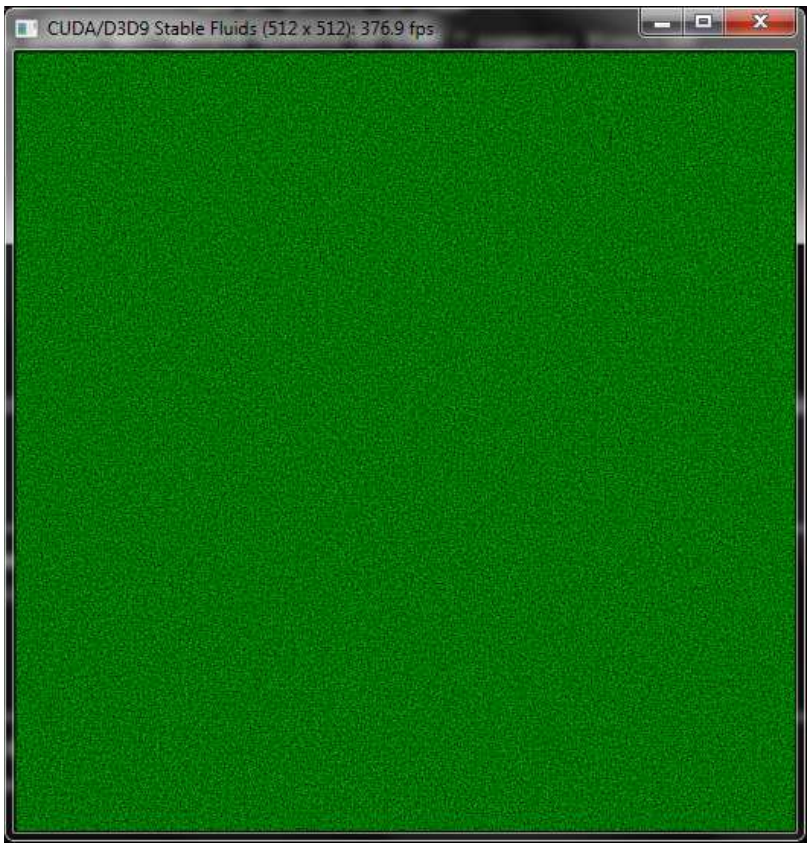


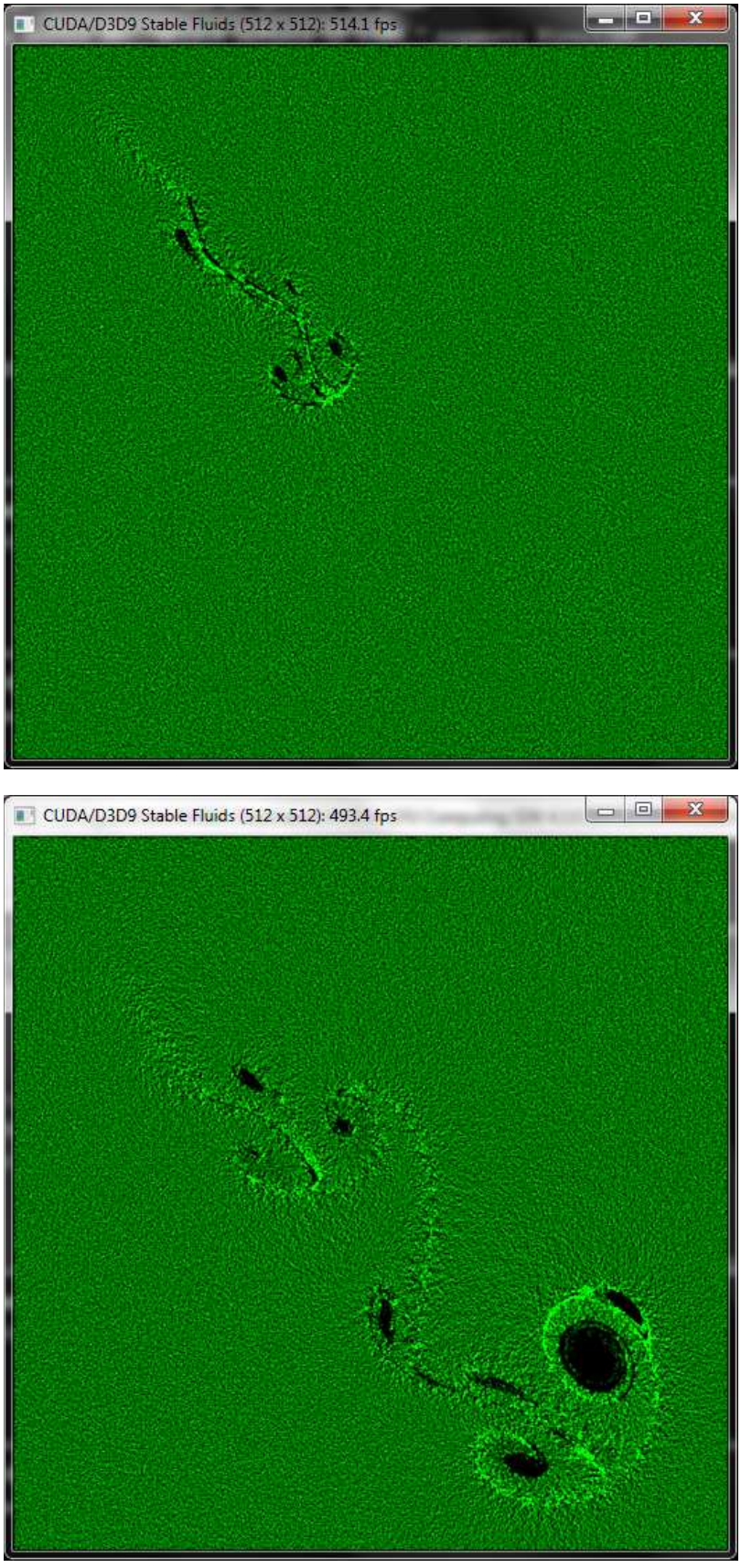
Décomposition de l'atmosphère en un maillage 2D



Décomposition de l'atmosphère en volumes élémentaires

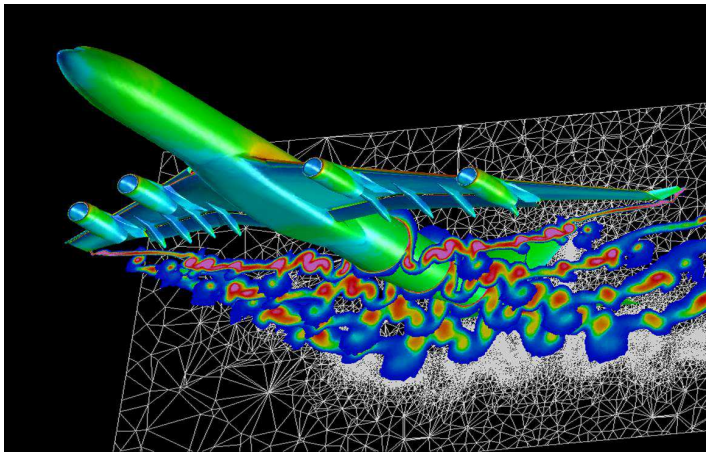
- Analyse des turbulences dans un fluide 2D
  - Problème en deux dimensions
  - Résolution des équations de mécanique des fluides (Navier-Stockes) complexe et extrêmement coûteuse en temps de calcul



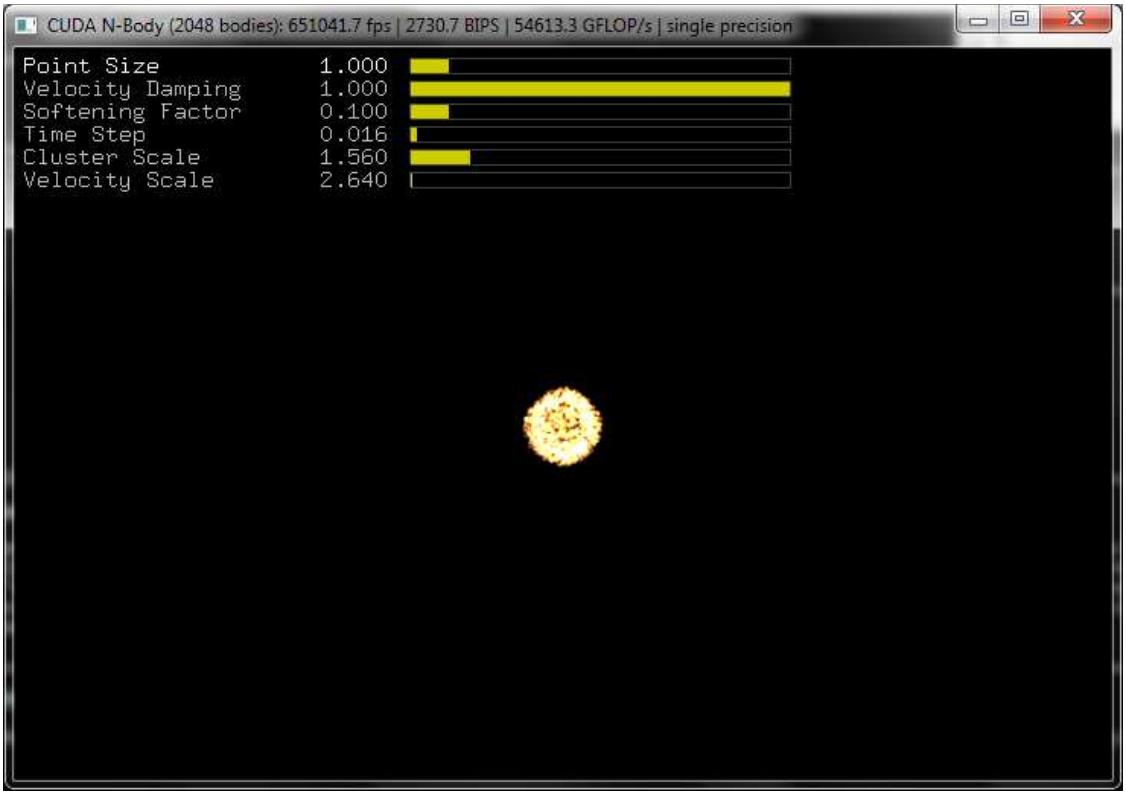


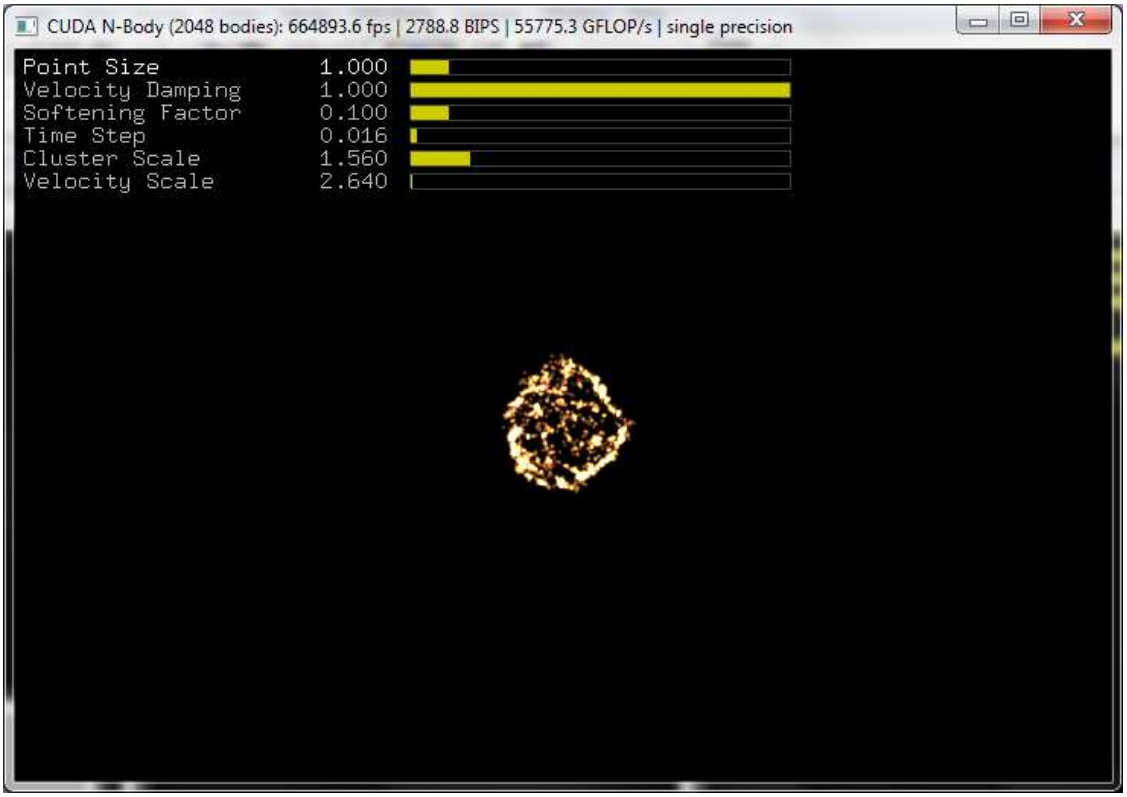
Simulation de fluides
Kit de développement CUDA (NVidia)

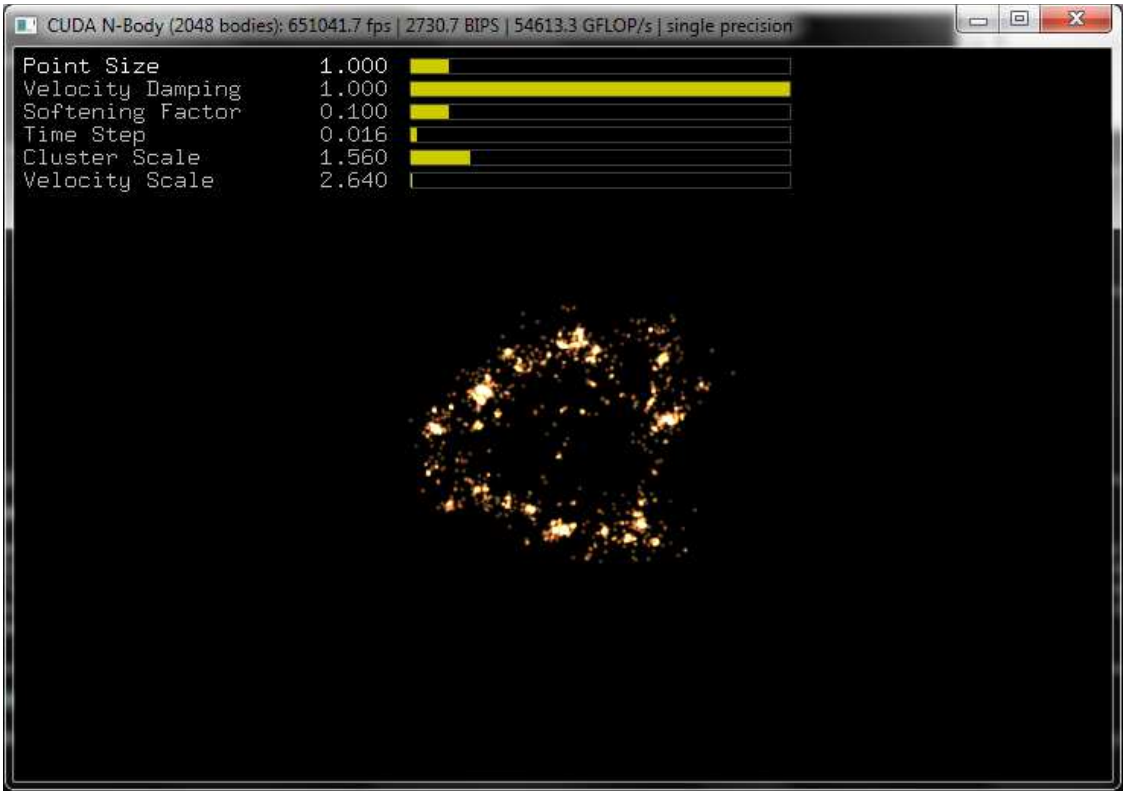
- Analyse des turbulences sur une aile d'avion
  - Problème en trois dimensions
  - Résolution des équations de mécanique des fluides (Navier-Stockes)
  - Historiquement, problème géré en dimension 2 avec simulation numérique sur des profils 2D de l'objet 3D
  - Accroissement de la puissance des ordinateurs -> Passage en 3D avec une aile complète
  - Bientôt possible de simuler l'écoulement sur un avion entier: ailes, fuselage, gouvernes, pylônes, réacteurs, ...

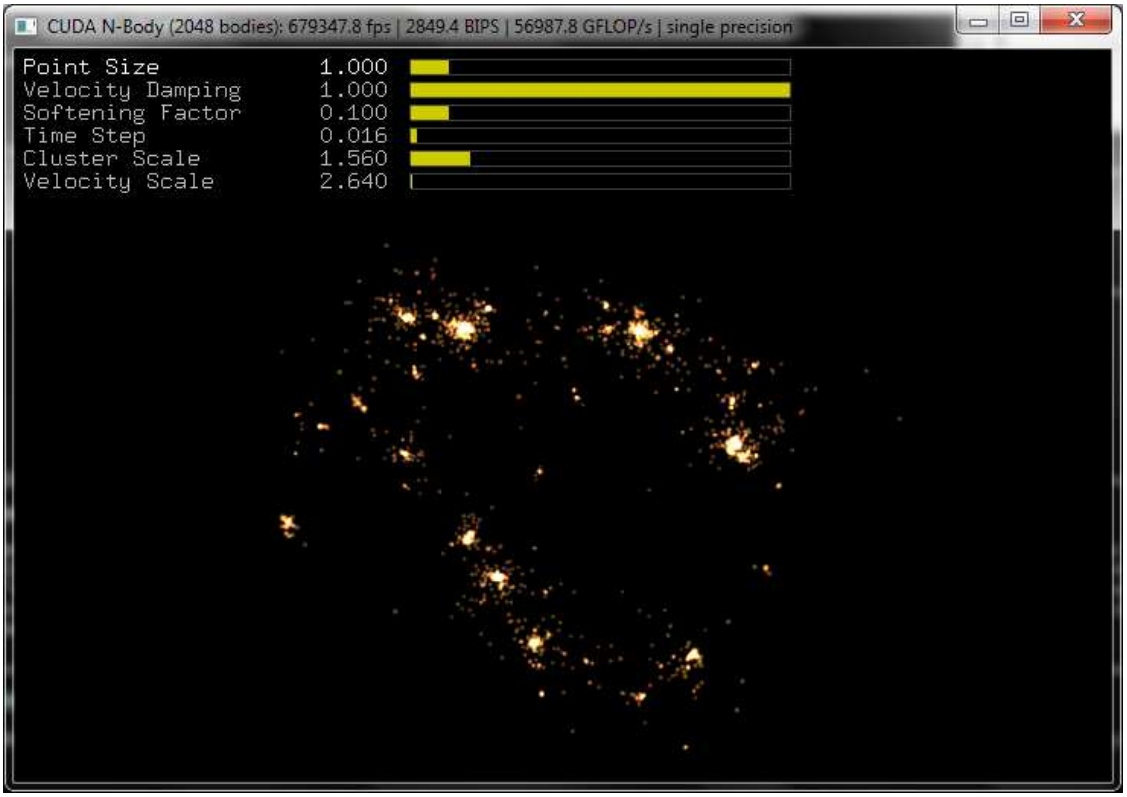


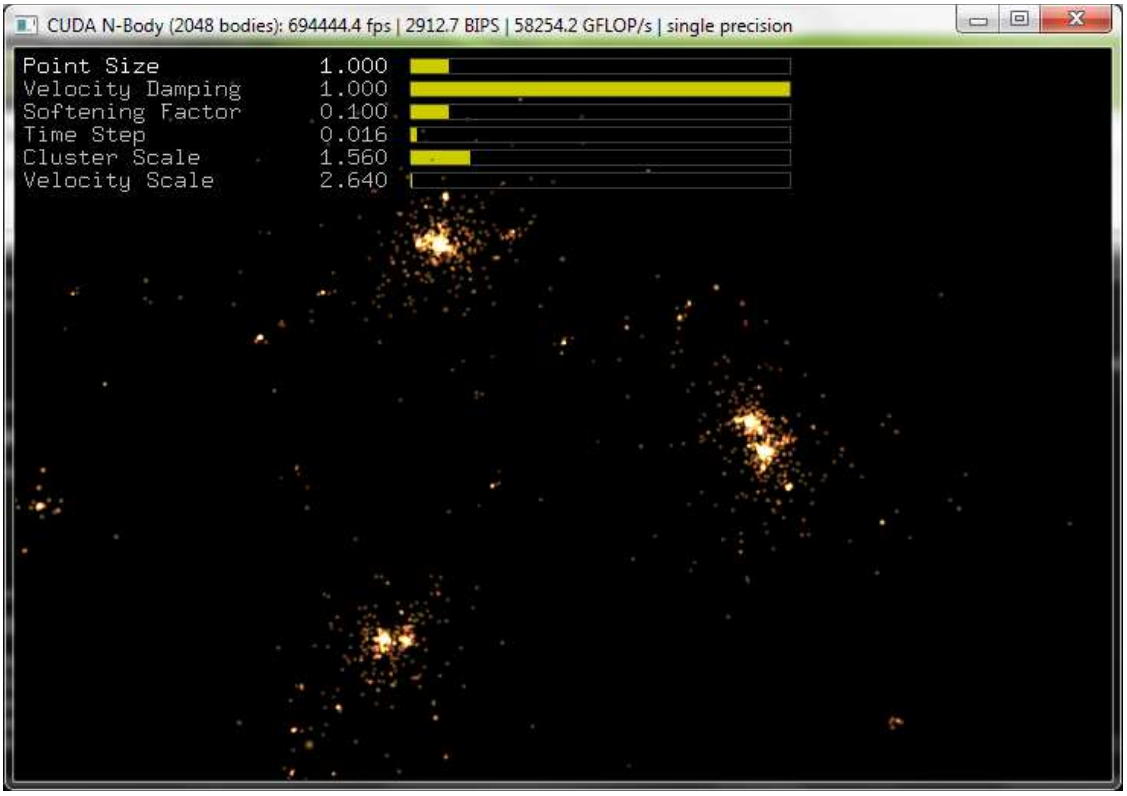
- Problème des n-corps
  - Comment évolue un système de n corps sous l'effet des forces de gravité qu'ils appliquent les uns sur les autres?
  - Pour n masses, 3*n équations différentielles du 2ème ordre
    -> 6*n inconnues



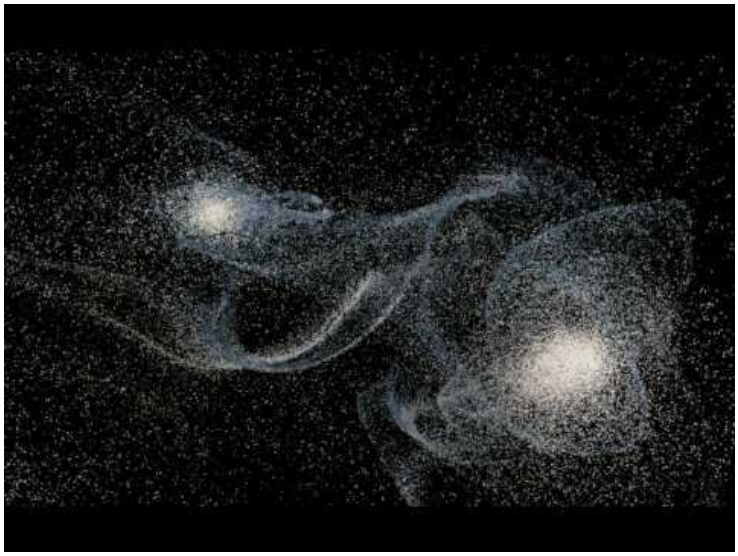

CUDA N-Body (2048 bodies): 664893.6 fps | 2788.8 BIPS | 55775.3 GFLOP/s | single precision
Point Size 1.000
Velocity Damping 1.000
Softening Factor 0.100
Time Step 0.016
Cluster Scale 1.560
Velocity Scale 2.640

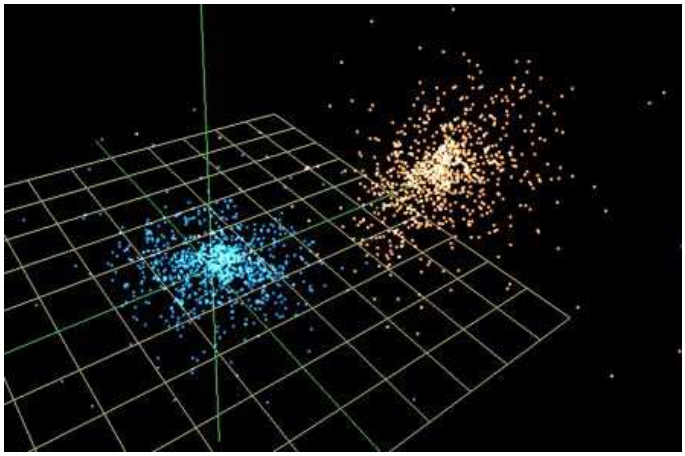

CUDA N-Body (2048 bodies): 651041.7 fps | 2730.7 BIPS | 54613.3 GFLOP/s | single precision
Point Size 1.000
Velocity Damping 1.000
Softening Factor 0.100
Time Step 0.016
Cluster Scale 1.560
Velocity Scale 2.640





Résolution du problème des n-corps
Kit de développement CUDA (NVidia)



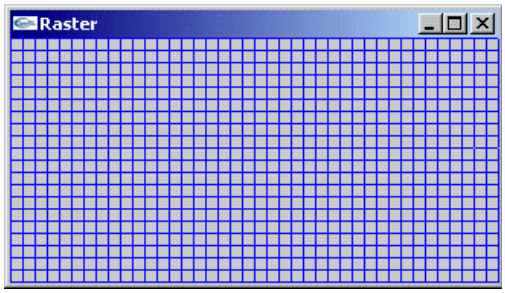


Simulations de collisions de galaxies

Matrices pour l'Informatique

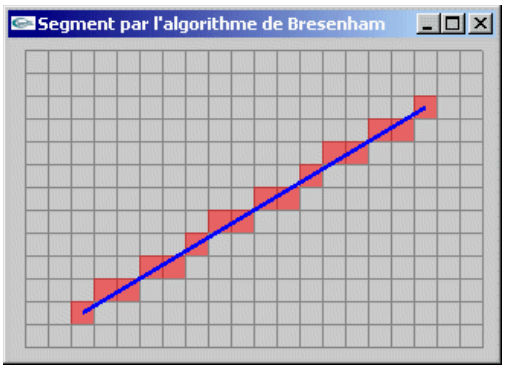
- Utilisation de matrices pour gérer les problèmes de l'informatique fondamentale qui se présentent naturellement sous forme 2D

Exemples

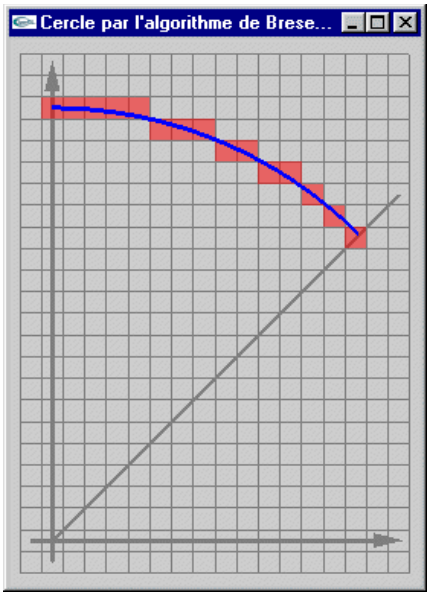
- Les écrans d'ordinateur
  - Technologie "raster" 

  - Une grille rectangulaire de pixels carrés pour lesquels on peut choisir individuellement la couleur
  - "Résolution": Nombres de colonnes et de lignes de pixels: 1024x768 -> 768 lignes de 1024 pixels
  - Autre caractéristique de la résolution: La "profondeur écran"
    - Nombre de bits de données affectés au stockage de la couleur d'un pixel
    - En 24 bits (3 octets), répartition en trois "teintes de base" stockées individuellement sur 8 bits
      -> 8 bits (1 octet) de composante rouge, 8 bits (1 octet) de composante verte et 8 bits (1 octet) de composante bleue
      -> Définition de chaque composante au moyen d'une valeur entière positive codée sur un octet (dans l'intervalle [0,255])
  - Couleurs classiques:
    - le noir (0,0,0)
    - le blanc (255,255,255)

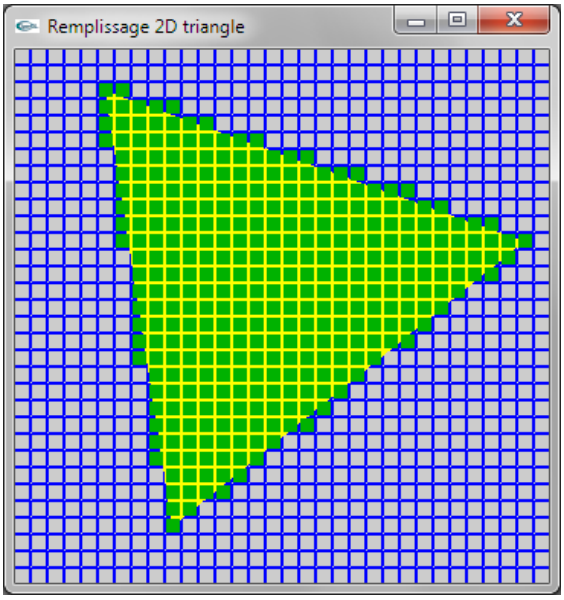
        - le rouge saturé (255,0,0)
        - le vert saturé (0,255,0)
        - le bleu saturé (0,0,255)
        - le jaune saturé (255,255,0)
        - le cyan saturé (0,255,255)
        - le magenta saturé (255,0,255)
    - Autres couleurs: Choix les valeurs "RVB" adéquates
    - Image affichée à l'écran: Une matrice de pixels codés en couleurs RVB stockée en mémoire
    - Opération d'affichage de plus bas-niveau: "Allumer" un pixel avec une couleur particulière
    - Autres opérations implantées en utilisant cette opération bas-niveau
    - Paramètres nécessaires à l'appel de fonction d'allumage de pixel:
        - Position en x: un entier compris dans l'intervalle [0,rx-1] où rx est la résolution d'affichage en x
        - Position en y: un entier compris dans l'intervalle [0,ry-1] où ry est la résolution d'affichage en y
        - Couleur d'affichage
    - "Librairie" de fonctions d'affichage graphique: Un grand nombre de fonctions de gestion de l'affichage en mode graphique et donc d'écriture dans la matrice de pixels:
        - Une ou plusieurs fonctions pour configurer la résolution d'affichage
        - Une ou plusieurs fonctions pour afficher un pixel selon une couleur
        - D'autres fonctions d'affichage plus élaboré:
            - Vidage la zone d'affichage avec une couleur (couleur de fond)
            - Traçage de segments de droite
            - Remplissage de polygones
            - ...



Dessin d'un segment de droite



Dessin d'un arc de cercle



Remplissage d'une facette triangulaire

```
/**
  * Logiciel de dessin bitmap.
  *
  * @author Nicolas JANEY, Universite de Franche-
Comte, France, <a href="mailto:nicolas.janey@univ-fcomte.fr">nicolas.janey@univ-
fcomte.fr</a>
  */

public class DessinBitmap {

   /**
    * Type structure de description d'une couleur
    */
   public static class Couleur {
     /**
     * Valeur de rouge.
     */
     public int r = 0;
     /**
     * Valeur de vert.
     */
     public int v = 0;
     /**
     * Valeur de bleu.
     */
     public int b = 0;
  }

   /**
    * Type structure de description d'une position 2D a coordonnees reelles
    */
   public static class Position2D {
     /**
     * Abscisse.
     */
     public double x = 0.0;
     /**
     * Ordonnee.
     */
     public double y = 0.0;
  }

   /**
    * Type structure de description d'un pixel
    */
```

```
  public static class Pixel {
    /**
    * Position.
    */
    public Position2D pos = new Position2D();
  }

  /**
   * Type structure de description d'un segment en 2D
   */
  public static class Segment2D {
    /**
    * Sommet initial.
    */
    public Position2D si = new Position2D();
    /**
    * Sommet final.
    */
    public Position2D sf = new Position2D();
  }

  /**
   * Type structure de description d'un rectangle en 2D
   */
  public static class Rectangle2D {
    /**
    * Position du centre.
    */
    public Position2D c = new Position2D();
    /**
    * Taille en x.
    */
    public double tx = 1.0;
    /**
    * Taille en y.
    */
    public double ty = 1.0;
    /**
    * Rotation.
    */
    public double rot = 0.0;
  }

  /**
   * Type structure de description d'un cercle en 2D
   */
  public static class Cercle2D {
    /**
    * Position du centre.
    */
    public Position2D c = new Position2D();
    /**
    * Rayon.
    */
    public double r = 1.0;
  }

  /**
   * Type structure de description d'une ellipse en 2D
   */
  public static class Ellipse2D {
    /**
    * Position du centre.
    */
    public Position2D c = new Position2D();
    /**
```

```
    * Demi grand axe en x.
    */
    public double a = 1.0;
    /**
    * Demi grand axe en y.
    */
    public double b = 1.0;
    /**
    * Rotation.
    */
    public double rot = 0.0;
}

 /**
  * Type structure de description d'un polygone regulier en 2D
  */
 public static class PolygoneRegulier2D {
    /**
    * Position du centre.
    */
    public Position2D c = new Position2D();
    /**
    * Nombre de sommets.
    */
    public int n = 3;
    /**
    * Rotation.
    */
    public double r = 1.0;
    /**
    * Rotation.
    */
    public double rot = 0.0;
}

 /**
  * Saisie clavier d'une couleur.
  *
  * @param message Le message d'accompagnement a la saisie.
  * @param c La couleur saisie.
  */
 public static void saisie(String message,Couleur c) {
  Ecran.afficherln(message);
  Ecran.afficher("Rouge : ");
  c.r = Clavier.saisirInt();
  Ecran.afficher("Vert  : ");
  c.v = Clavier.saisirInt();
  Ecran.afficher("Bleu  : ");
  c.b = Clavier.saisirInt();
}

 /**
  * Saisie clavier d'une position en 2D.
  *
  * @param message Le message d'accompagnement a la saisie.
  * @param p La position saisie.
  */
 public static void saisie(String message,Position2D p) {
  Ecran.afficherln(message);
  Ecran.afficher("x : ");
  p.x = Clavier.saisirInt();
  Ecran.afficher("y : ");
  p.y = Clavier.saisirInt();
}

 /**
```

```
    * Saisie clavier d'un pixel.
    *
    * @param message Le message d'accompagnement a la saisie.
    * @param p Le pixel saisi.
    */
   public static void saisie(String message,Pixel p) {
    Ecran.afficherln(message);
    saisie("Position",p.pos);
  }

   /**
    * Saisie clavier d'un segment 2D.
    *
    * @param message Le message d'accompagnement a la saisie.
    * @param s Le segment 2D saisi.
    */
   public static void saisie(String message,Segment2D s) {
    Ecran.afficherln(message);
    saisie("Sommet initial",s.si);
    saisie("Sommet final",s.sf);
  }

   /**
    * Saisie clavier d'un rectangle 2D.
    *
    * @param message Le message d'accompagnement a la saisie.
    * @param r Le rectangle 2D saisi.
    */
   public static void saisie(String message,Rectangle2D r) {
    Ecran.afficherln(message);
    saisie("Centre",r.c);
    Ecran.afficher("Largeur  : ");
    r.tx = Clavier.saisirInt();
    Ecran.afficher("Hauteur  : ");
    r.ty = Clavier.saisirInt();
    Ecran.afficher("Rotation : ");
    r.rot = Clavier.saisirDouble();
  }

   /**
    * Saisie clavier d'un cercle 2D.
    *
    * @param message Le message d'accompagnement a la saisie.
    * @param c Le cercle 2D saisi.
    */
   public static void saisie(String message,Cercle2D c) {
    Ecran.afficherln(message);
    saisie("Centre",c.c);
    Ecran.afficher("Rayon : ");
    c.r = Clavier.saisirDouble();
  }

   /**
    * Saisie clavier d'une ellipse 2D.
    *
    * @param message Le message d'accompagnement a la saisie.
    * @param e L'ellipse 2D saisie.
    */
   public static void saisie(String message,Ellipse2D e) {
    Ecran.afficherln(message);
    saisie("Centre",e.c);
    Ecran.afficher("Largeur  : ");
    e.a = Clavier.saisirInt();
    Ecran.afficher("Hauteur  : ");
    e.b = Clavier.saisirInt();
    Ecran.afficher("Rotation : ");
```

```
    e.rot = Clavier.saisirDouble();
  }

   /**
    * Saisie clavier d'un polygone regulier 2D.
    *
    * @param message Le message d'accompagnement a la saisie.
    * @param p Le polygone regulier 2D saisi.
    */
   public static void saisie(String message,PolygoneRegulier2D p) {
    Ecran.afficherln(message);
    saisie("Centre",p.c);
    Ecran.afficher("Rayon          : ");
    p.r = Clavier.saisirDouble();
    Ecran.afficher("Nombre sommets : ");
    p.n = Clavier.saisirInt();
    Ecran.afficher("Rotation       : ");
    p.rot = Clavier.saisirDouble();
  }

   /**
    * Calcul de la position obtenue par rotation d'une position autour d'une position.
    *
    * @param c Le centre de rotation.
    * @param x L'abscisse de la position a traiter par rapport au centre de rotation.
    * @param y L'ordonnee de la position a traiter par rapport au centre de rotation.
    * @param a L'angle de rotation (en radians).
    * @return La Position2D calculee.
    */
   public static Position2D positionAvecRotation
(Position2D c,double x,double y,double a) {
    Position2D p = new Position2D();
    p.x = c.x + x*Math.cos(a)-y*Math.sin(a);
    p.y = c.y + x*Math.sin(a)+y*Math.cos(a);
     return p;
  }

   /**
    * Calcul de la suite de sommets d'un rectangle 2D.
    *
    * @param r Le rectangle 2D.
    * @return Le tableau de Position2D calcule.
    */
   public static Position2D [] sommets(Rectangle2D r) {
    Position2D [] t = new Position2D[4];
    t[0] = positionAvecRotation(r.c,r.tx/2.0,r.ty/2.0,r.rot);
    t[1] = positionAvecRotation(r.c,-r.tx/2.0,r.ty/2.0,r.rot);
    t[2] = positionAvecRotation(r.c,-r.tx/2.0,-r.ty/2.0,r.rot);
    t[3] = positionAvecRotation(r.c,r.tx/2.0,-r.ty/2.0,r.rot);
     return t;
  }

   /**
    * Calcul de la suite de sommets d'un cercle 2D.
    *
    * @param c Le cercle 2D.
    * @param n Le nombre de sommets a calculer.
    * @return Le tableau de Position2D calcule.
    */
   public static Position2D [] sommets(Cercle2D c,int n) {
    Position2D [] t = new Position2D[n];
     for ( int i = 0 ; i < n ; i++ ) {
       double a = i*2.0*Math.PI/n;
      t[i] = new Position2D();
      t[i].x = c.c.x+c.r*Math.cos(a);
      t[i].y = c.c.y+c.r*Math.sin(a); }
```

```
    return t;
  }

  /**
   * Calcul de la suite de sommets d'une ellipse 2D.
   *
   * @param e L'ellipse 2D.
   * @param n Le nombre de sommets a calculer.
   * @return Le tableau de Position2D calcule.
   */
  public static Position2D [] sommets(Ellipse2D e,int n) {
   Position2D [] t = new Position2D[n];
    for ( int i = 0 ; i < n ; i++ ) {
      double a = i*2.0*Math.PI/n;
     t[i] = positionAvecRotation(e.c,e.a*Math.cos(a),e.b*Math.sin(a),e.rot); }
    return t;
  }

  /**
   * Calcul de la suite de sommets d'un polygone regulier 2D.
   *
   * @param p Le polygone regulier 2D.
   * @return Le tableau de Position2D calcule.
   */
  public static Position2D [] sommets(PolygoneRegulier2D p) {
   Position2D [] t = new Position2D[p.n];
    for ( int i = 0 ; i < p.n ; i++ ) {
      double a = i*2.0*Math.PI/p.n;
     t[i] = positionAvecRotation(p.c,p.r*Math.cos(a),p.r*Math.sin(a),p.rot); }
    return t;
  }

  /**
   * Affichage en boucle fermee de segements de droite d'un tableau de position 2D.
   *
   * @param t Le tableau de positions 2D a afficher graphiquement.
   */
  public static void affichageTableauSommets(Position2D [] t) {
    for ( int i = 0 ; i < t.length ; i++ ) {
     EcranGraphique.drawLine((int) t[i].x,(int) t[i].y,
                             (int) t[(i+1)%t.length].x,(int) t[(i+1)%t.length].y); }
  }

  /**
   * Affichage graphique d'un pixel.
   *
   * @param p Le pixel a afficher.
   * @param c La couleur d'affichage.
   */
  public static void affichageGraphique(Pixel p,Couleur c) {
   EcranGraphique.setColor(c.r,c.v,c.b);
   EcranGraphique.drawPixel((int) p.pos.x,(int) p.pos.y);
  }

  /**
   * Affichage graphique d'un segment 2D.
   *
   * @param s Le segment 2D a afficher.
   * @param c La couleur d'affichage.
   */
  public static void affichageGraphique(Segment2D s,Couleur c) {
   EcranGraphique.setColor(c.r,c.v,c.b);
   EcranGraphique.drawLine((int) s.si.x,(int) s.si.y,(int) s.sf.x,(int) s.sf.y);
  }

  /**
```

```
    * Affichage graphique d'un rectangle 2D.
    *
    * @param r Le rectangle 2D a afficher.
    * @param c La couleur d'affichage.
    */
   public static void affichageGraphique(Rectangle2D r,Couleur c) {
    EcranGraphique.setColor(c.r,c.v,c.b);
    affichageTableauSommets(sommets(r));
  }

   /**
    * Affichage graphique d'un cercle 2D.
    *
    * @param cl Le cercle 2D a afficher.
    * @param c La couleur d'affichage.
    */
   public static void affichageGraphique(Cercle2D cl,Couleur c) {
    EcranGraphique.setColor(c.r,c.v,c.b);
    affichageTableauSommets(sommets(cl,180));
  }

   /**
    * Affichage graphique d'une ellipse 2D.
    *
    * @param e L'ellipse 2D a afficher.
    * @param c La couleur d'affichage.
    */
   public static void affichageGraphique(Ellipse2D e,Couleur c) {
    EcranGraphique.setColor(c.r,c.v,c.b);
    affichageTableauSommets(sommets(e,180));
  }

   /**
    * Affichage graphique d'un polygone regulier 2D.
    *
    * @param p Le polygone regulier 2D a afficher.
    * @param c La couleur d'affichage.
    */
   public static void affichageGraphique(PolygoneRegulier2D p,Couleur c) {
    EcranGraphique.setColor(c.r,c.v,c.b);
    affichageTableauSommets(sommets(p));
  }

   /**
    * Affichage graphique d'une croix.
    *
    * @param x L'abscisse du centre de la croix.
    * @param y L'ordonnee du centre de la croix.
    */
   public static void croix(int x,int y,Cercle2D cl,Couleur c) {
    EcranGraphique.setColor(c.r,c.v,c.b);
    EcranGraphique.drawLine(x-10,y,x+10,y);
    EcranGraphique.drawLine(x,y-10,x,y+10);
    affichageGraphique(cl,c);
  }

   /**
    * Saisie d'un cercle 2D interactivement a la souris.
    *
    * @param cl Le cercle a saisir interactivement.
    */
   public static void saisieInteractiveSouris(Cercle2D cl) {
     while ( EcranGraphique.getMouseState() == 0 );
    EcranGraphique.setXorMode(true);
     int xOld = EcranGraphique.getMouseX();
     int yOld = EcranGraphique.getMouseY();
```

```
    Couleur blanc = new Couleur();
   blanc.r = 255;
   blanc.v = 255;
   blanc.b = 255;
   cl.c.x = xOld;
   cl.c.y = yOld;
   cl.r = 0.0;
   croix(xOld,yOld,cl,blanc);
    while ( EcranGraphique.getMouseState() == 1 ) {
      int x = EcranGraphique.getMouseX();
      int y = EcranGraphique.getMouseY();
      if ( ( x != xOld ) || ( y != yOld ) ) {
       croix(xOld,yOld,cl,blanc);
       xOld = x;
       yOld = y;
       cl.r = Math.sqrt(Math.pow(x-cl.c.x,2.0)+Math.pow(y-cl.c.y,2.0));
       croix(x,y,cl,blanc);
     EcranGraphique.flush(); } }
   croix(xOld,yOld,cl,blanc);
   EcranGraphique.flush();
   EcranGraphique.setXorMode(false);
 }

  /**
   * Gestion du menu principal de choix de l'action a realiser.
   *
   * @return L'entier saisi.
   */
  public static int resultatMenuPrincipal() {
    int v;
   Ecran.sautDeLigne();
   Ecran.afficherln(" 1: Choix de la couleur de trace");
   Ecran.afficherln(" 2: Afficher un pixel");
   Ecran.afficherln(" 3: Afficher un segment de droite");
   Ecran.afficherln(" 4: Afficher un rectangle");
   Ecran.afficherln(" 5: Afficher un cercle");
   Ecran.afficherln(" 6: Afficher une ellipse");
   Ecran.afficherln(" 7: Afficher un polygone regulier");
   Ecran.afficherln(" 8: Afficher n objets tires au hasard");
   Ecran.afficherln(" 9: Choisir la couleur d'effacement de la fenetre");
   Ecran.afficherln("10: Effacer la fenetre");
   Ecran.afficherln("11: Tracer dynamiquement un cercle");
   Ecran.afficherln(" 0: Quitter");
   Ecran.afficher("Que voulez vous faire ? ");
   v = Clavier.saisirInt();
   Ecran.sautDeLigne();
    return v;
 }

  /**
   * Programme principal.
   *
   * @param args Le tableau des parametres de lancement.
   */

  public static void main(String [] args) {
    int n;
    int tx;
    int ty;
    int choix;
   Pixel px = new Pixel();
   Segment2D sg = new Segment2D();
   Rectangle2D rt = new Rectangle2D();
   Cercle2D cl = new Cercle2D();
   Ellipse2D el = new Ellipse2D();
   PolygoneRegulier2D pr = new PolygoneRegulier2D();
```

```
/* Couleurs de trace et d'effacement                */
  Couleur ct = new Couleur();
 ct.r = 0;
 ct.v = 0;
 ct.b = 0;
  Couleur ce = new Couleur();
 ce.r = 255;
 ce.v = 255;
 ce.b = 255;
/* Initialisation de la fenetre d'affichage         */
 Ecran.afficherln("Taille de la fenetre");
 Ecran.afficher("Tx : ");
 tx = Clavier.saisirInt();
 Ecran.afficher("Ty : ");
 ty = Clavier.saisirInt();
 EcranGraphique.init(50,50,tx,ty,tx-40,ty-80,"Dessin bitmap");
 EcranGraphique.setClearColor(255,255,255);
 EcranGraphique.clear();
 EcranGraphique.flush();
 choix = resultatMenuPrincipal();
  while ( choix != 0 ) {
    switch (choix) {
      case 1 :
       saisie("Couleur de trace?",ct);
        break;
      case 2 :
       saisie("Votre pixel?",px);
       affichageGraphique(px,ct);
       EcranGraphique.flush();
        break;
      case 3 :
       saisie("Votre segment?",sg);
       affichageGraphique(sg,ct);
       EcranGraphique.flush();
        break;
      case 4 :
       saisie("Votre rectangle?",rt);
       affichageGraphique(rt,ct);
       EcranGraphique.flush();
        break;
      case 5 :
       saisie("Votre cercle?",cl);
       affichageGraphique(cl,ct);
       EcranGraphique.flush();
        break;
      case 6 :
       saisie("Votre ellipse?",el);
       affichageGraphique(el,ct);
       EcranGraphique.flush();
        break;
      case 7 :
       saisie("Votre polygone regulier?",pr);
       affichageGraphique(pr,ct);
       EcranGraphique.flush();
        break;
      case 8 :
       Ecran.afficher("Nombre d'objets : ");
       n = Clavier.saisirInt();
        for ( int i = 0 ; i < n ; i++ ) {
          int type =(int) (Math.random()*6.0);
          switch(type) {
            case 0 :
             px.pos.x = Math.random()*tx;
             px.pos.y = Math.random()*ty;
             affichageGraphique(px,ct);
              break;
```

```
                case 1 :
                 sg.si.x = Math.random()*tx;
                 sg.si.y = Math.random()*ty;
                 sg.sf.x = Math.random()*tx;
                 sg.sf.y = Math.random()*ty;
                 affichageGraphique(sg,ct);
                  break;
                case 2 :
                 rt.c.x = Math.random()*tx;
                 rt.c.y = Math.random()*ty;
                 rt.tx = 3.0+Math.random()*500.0;
                 rt.ty = 3.0+Math.random()*500.0;
                 rt.rot = Math.random()*2.0*Math.PI;
                 affichageGraphique(rt,ct);
                  break;
                case 3 :
                 cl.c.x = Math.random()*tx;
                 cl.c.y = Math.random()*ty;
                 cl.r = 3.0+Math.random()*200.0;
                 affichageGraphique(cl,ct);
                  break;
                case 4 :
                 el.c.x = Math.random()*tx;
                 el.c.y = Math.random()*ty;
                 el.a = 3.0+Math.random()*200.0;
                 el.b = 3.0+Math.random()*200.0;
                 el.rot = Math.random()*2.0*Math.PI;
                 affichageGraphique(el,ct);
                  break;
                case 5 :
                 pr.c.x = Math.random()*tx;
                 pr.c.y = Math.random()*ty;
                 pr.n = 3+(int) (Math.random()*10.0);
                 pr.r = 3.0+Math.random()*200.0;
                 pr.rot = Math.random()*2.0*Math.PI;
                 affichageGraphique(pr,ct);
                  break; } }
          EcranGraphique.flush();
           break;
         case 9 :
          saisie("Couleur d'effacement?",ce);
           break;
         case 10 :
          EcranGraphique.setClearColor(ce.r,ce.v,ce.b);
          EcranGraphique.clear();
          EcranGraphique.flush();
           break;
         case 11 :
          saisieInteractiveSouris(cl);
          affichageGraphique(cl,ct);
          EcranGraphique.flush();
           break; }
       choix = resultatMenuPrincipal(); }
    Ecran.afficherln("Return pour quitter");
    Clavier.saisirString();
    EcranGraphique.exit();
  }
}
```

Exemple d'affichage graphique bitmap

- Deux techniques de stockage d'images dans des fichiers:
  - Vectoriel: Images stockées sous la forme d'objets graphiques de type segment de droite, rectangle, rectangle rempli, cercle, disque, ...

- Bitmap: Images stockées sous la forme d'une grille rectangulaire de pixels décrits individuellement par une couleur

- Déclinaison de ces techniques de diverses manières avec pour conséquence l'apparition de divers formats de fichiers:
  - Exemples de formats vectoriels: Postscript (.ps) et SVG (.svg, Scalable Vector Graphic)
  - Exemples de formats bitmap: JPEG, BMP, GIF, PNG
    - Compression ou non
    - Si compression, perte de qualité ou non
    - Format 8 bits, 24 bits ou 32 bits
    - ...

- Traitement particulier applicable aux images bitmaps: Réalisation d'opérations de traitement d'image visant à en changer les caractéristiques:
  - Afficher en fausses couleurs
  - Augmenter/diminuer la luminosité
  - Augmenter/diminuer le contraste
  - Flouter
  - Extraire les contours
  - Négativer
  - ...

  - Traitement mathématique de chaque pixel de la matrice de pixels représentant l'image
    - Calcul de la nouvelle valeur d'un pixel à partir de son ancienne valeur
      -> Filtrage spectral
    - Calcul de la nouvelle valeur d'un pixel à partir de son ancienne valeur ainsi que des valeurs d'autres pixels (généralement les pixels voisins)
      -> Filtrage matriciel

Image originale

Image floutée
Remplacement de chaque pixel par la moyenne de ce pixel et des pixels voisins

Image après extraction de contours
Gradiant calculé sur les pixels voisins

```
/**
* Conversion d'une couleur RVB en un entier.
*
* @param r La composante de rouge de la couleur a convertir.
* @param v La composante de vert de la couleur a convertir.
* @param b La composante de bleu de la couleur a convertir.
*
* @return L'entier codant la Couleur
* selon la formule b*256*256+v*256+r.
*/
   public static int couleur(int r,int v,int b) {
   int couleur;
  couleur = b*256*256+v*256+r;
   return couleur;
}

/**
* Diminution de la luminosite d'une image.
*
* @param img Le tableau image a traiter.
* @param prct Le pourcentage de modification.
*/
static void diminuerLuminosite(int [][] img,double prct) {
   double fact = 1.0-prct/100.0;
   int r;
   int v;
```

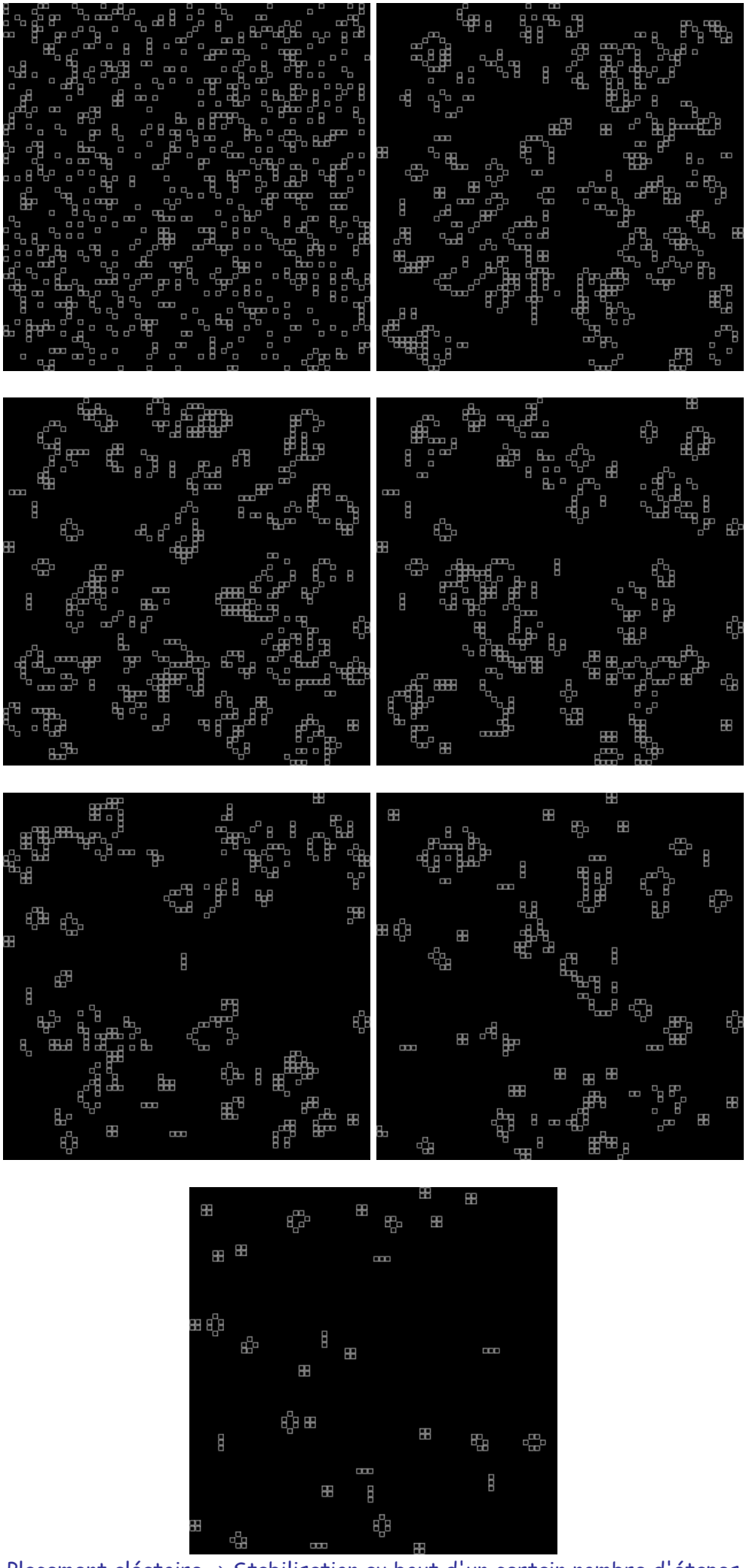
```
   int b;
   for ( int x = 0 ; x < img.length ; x++ ) {
     for ( int y = 0 ; y < img[0].length ; y++ ) {
      r = img[x][y]%256;
      v = (img[x][y]/256)%256;
      b = img[x][y]/65536;
      r =(int) (r*fact);
      v =(int) (v*fact);
      b =(int) (b*fact);
      img[x][y] = couleur(r,v,b); } }
}

/**
* Augmentation de la luminosite d'une image.
*
* @param img Le tableau image a traiter.
* @param prct Le pourcentage de modification.
*/
static void augmenterLuminosite(int [][] img,double prct) {
   double fact = 1.0-prct/100.0;
   int r;
   int v;
   int b;
   for ( int x = 0 ; x < img.length ; x++ ) {
     for ( int y = 0 ; y < img[0].length ; y++ ) {
      r = img[x][y]%256;
      v = (img[x][y]/256)%256;
      b = img[x][y]/65536;
      r = 255 - (int) ((255-r)*fact);
      v = 255 - (int) ((255-v)*fact);
      b = 255 - (int) ((255-b)*fact);
      img[x][y] = couleur(r,v,b); } }
}

/**
* Diminution du contraste d'une image.
*
* @param img Le tableau image a traiter.
* @param prct Le pourcentage de modification.
*/
static void diminuerContraste(int [][] img,double prct) {
   double fact = 1.0-prct/100.0;
   int r;
   int v;
   int b;
   for ( int x = 0 ; x < img.length ; x++ ) {
     for ( int y = 0 ; y < img[0].length ; y++ ) {
      r = img[x][y]%256;
      v = (img[x][y]/256)%256;
      b = img[x][y]/65536;
      r = 127 -(int) ((127-r)*fact);
      v = 127 -(int) ((127-v)*fact);
      b = 127 -(int) ((127-b)*fact);
      img[x][y] = couleur(r,v,b); } }
}

/**
* Augmentation du contraste d'une image.
*
* @param img Le tableau image a traiter.
* @param prct Le pourcentage de modification.
*/
static void augmenterContraste(int [][] img,double prct) {
   double fact = 1.0-prct/100.0;
   int r;
   int v;
```

```
   int b;
   for ( int x = 0 ; x < img.length ; x++ ) {
     for ( int y = 0 ; y < img[0].length ; y++ ) {
      r = img[x][y]%256;
      v = (img[x][y]/256)%256;
      b = img[x][y]/65536;
       if ( r <= 127 )
        r =(int) (r*fact);
         else
        r = 255 - (int) ((255-r)*fact);
       if ( v <= 127 )
        v =(int) (v*fact);
         else
        v = 255 - (int) ((255-v)*fact);
       if ( b <= 127 )
        b =(int) (b*fact);
         else
        b = 255 - (int) ((255-b)*fact);
      img[x][y] = couleur(r,v,b); } }
}

/* Methode de clonage d'un tableau          */
/* a 2 indices de int                       */

static int [][] clone(int [][] t) {
   int i;
   int j;
   int n = t.length;
   int m = t[0].length;
   int [][] nt = new int[n][m];
   for ( i = 0 ; i < n ; i = i+1 ) {
     for ( j = 0 ; j < m ; j = j+1 ) {
      nt[i][j] = t[i][j]; } }
   return nt;
}

/**
* Floutage d'une image.
*
* @param img Le tableau image a traiter.
*/
static void flouter(int [][] img) {
   int [][] image = clone(img);
   int r;
   int v;
   int b;
   int totr;
   int totv;
   int totb;
   int nb;
   for ( int x = 0 ; x < image.length ; x++ ) {
     for ( int y = 0 ; y < image[0].length ; y++ ) {
      totr = 0;
      totv = 0;
      totb = 0;
      nb = 0;
       for ( int xx = x-2 ; xx <= x+2 ; xx++ ) {
         for ( int yy = y-2 ; yy <= y+2 ; yy++ ) {
           if ( ( xx >= 0 ) && ( xx < image.length ) &&
                ( yy >= 0 ) && ( yy < image[0].length ) ) {
            totr = totr+image[xx][yy]%256;
            totv = totv+(image[xx][yy]/256)%256;
            totb = totb+image[xx][yy]/65536;
            nb = nb+1; } } }
      r = totr/nb;
      v = totv/nb;
```

```
      b = totb/nb;
      img[x][y] = couleur(r,v,b); } }
}

/**
* Extraction de contours dans une image.
*
* @param img Le tableau image a traiter.
*/
static void extraireContours(int [][] img) {
   int [][] image = clone(img);
   int r;
   int v;
   int b;
   for ( int x = 0 ; x < image.length ; x++ ) {
    img[x][0] = 0;
    img[x][img[0].length-1] = 0; }
   for ( int y = 1 ; y < image[0].length-1 ; y++ ) {
    img[0][y] = 0;
    img[img.length-1][y] = 0; }
   for ( int x = 1 ; x < image.length-1 ; x++ ) {
     for ( int y = 1 ; y < image[0].length-1 ; y++ ) {
      r = 4*(image[x][y]%256);
      v = 4*((image[x][y]/256)%256);
      b = 4*(image[x][y]/65536);
      r = r-image[x-1][y]%256;
      v = v-(image[x-1][y]/256)%256;
      b = b-image[x-1][y]/65536;
      r = r-image[x+1][y]%256;
      v = v-(image[x+1][y]/256)%256;
      b = b-image[x+1][y]/65536;
      r = r-image[x][y-1]%256;
      v = v-(image[x][y-1]/256)%256;
      b = b-image[x][y-1]/65536;
      r = r-image[x][y+1]%256;
      v = v-(image[x][y+1]/256)%256;
      b = b-image[x][y+1]/65536;
      r = Math.abs(r);
       if ( r > 255 )
        r = 255;
      v = Math.abs(v);
       if ( v > 255 )
        v = 255;
      b = Math.abs(b);
       if ( b > 255 )
        b = 255;
      img[x][y] = couleur(r,v,b); } }
}
```

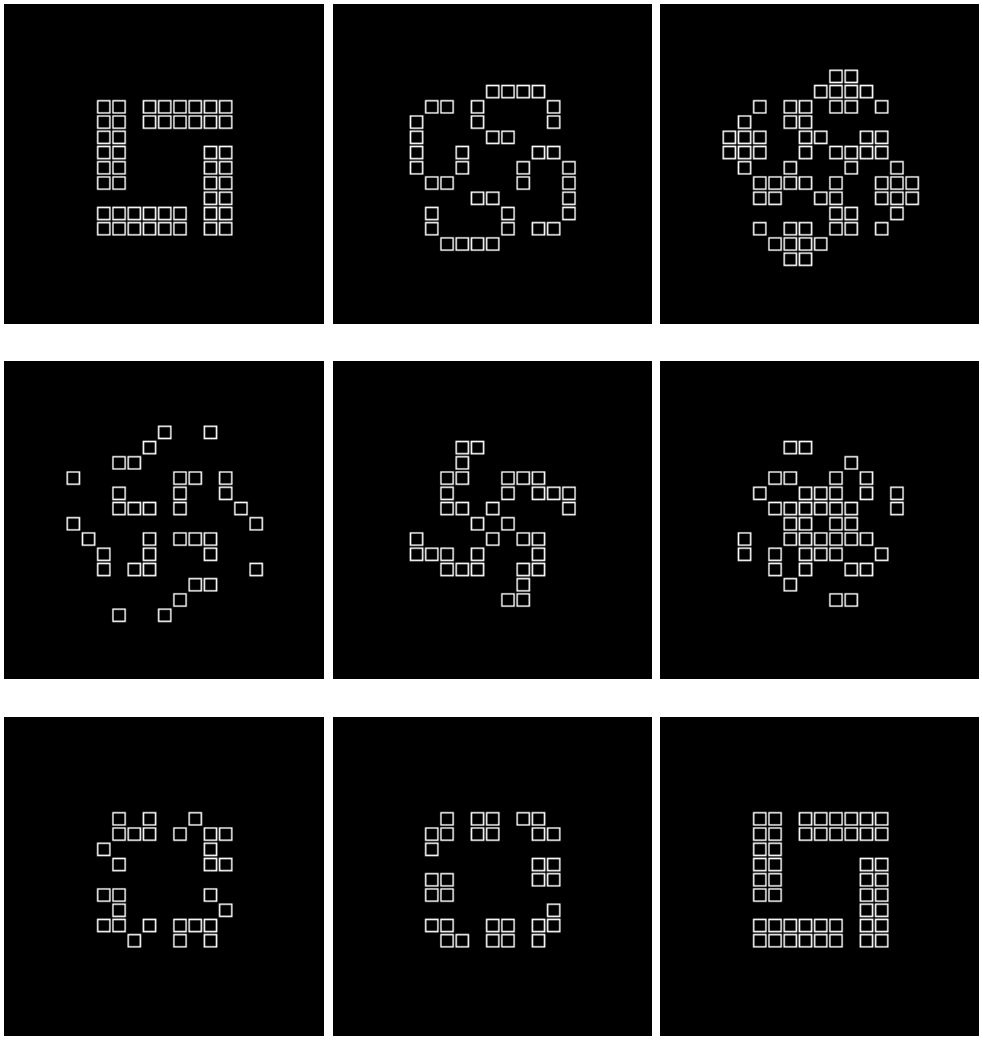
Exemple de traitement d'image

- Jeu de la vie
  - Une grille de booléens initialisés à vrai ou à faux
  - Chaque cellule évolue en fonction du contenu de ses 8 voisines
    - Si 2 voisins à vrai, la cellule ne change pas
    - Si 3 voisins à vrai, la cellule passe à vrai
    - Dans tous les autres cas, la cellule passe à faux
- Automate cellulaire
- Evolution d'un eco-système

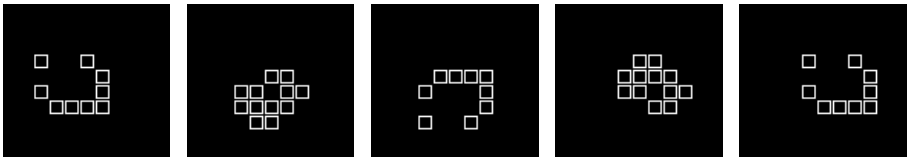


Placement aléatoire -> Stabilisation au bout d'un certain nombre d'étapes

Exemple d'exécution



Evolution cyclique en place: Galaxie
Exemple d'exécution



Evolution cyclique avec déplacement régulier: Vaisseau
Exemple d'exécution

- Jeux 2D

Tétris

Candy Crush

Auteur: Nicolas JANEY
UFR Sciences et Techniques
Université de Besançon
16 Route de Gray, 25030 Besançon
nicolas.janey@univ-fcomte.fr