Simulation physique de textiles sur grappe de processeurs

Florence Zara

ID-IMAG [1]
51 avenue Jean Kuntzmann
38330 MONTBONNOT SAINT-MARTIN

Florence.Zara@imag.fr

Résumé : *Dans le cadre de l'animation d'objets 3D en synthèse d'image, la simulation de textiles pour représenter des personnages habillés est en plein essort. Les éléments fondamentales de la physique, comme la vitesse, les forces (gravitation, ...), sont alors employés pour modéliser le mouvement de plusieurs objets interagissants dans un souci de réalisme. Le but de nos travaux est la diminution du temps de calcul d'une image de personnage afin d'obtenir des animations dynamiques temps réel. Ce papier décrit une simulation de tissu basée sur une modélisation physique des particules, avec une intégration implicite du temps, et une résolution linéaire par un gradient conjugué par blocs parallèles. Les algorithmes de la simulation ont été parallélisés et son exécution est effectuée sur une grappe de PC.*

Mots-clés : Simulation, Textiles, Modèles physiques, Parallélisme, Grappe de PC, Temps réel.

1 Introduction

Depuis longtemps, l'animation de textiles virtuels constitue un axe de recherche particulièrement important au sein de la communauté de l'informatique graphique. Elle est d'autant plus privilégiée qu'elle connait de nombreuses applications dans le monde industriel. Une des difficultés principales de notre simulation de textiles est le temps réel ($\simeq$ 25 Hz) en utilisant des modèles physiques pour la modélisation des mouvements. C'est-à-dire que nous devons chercher absolument à minimiser la complexité du système, ainsi que le temps d'exécution de notre application. L'obtention d'une simulation de textiles dynamiques temps réel intéresse aussi bien le monde industriel (industries du textile), que le monde du loisir. Au niveau industriel, cette simulation intégrée aux logiciels de CAO permettrait de réduire considérablement les coûts de mise au point en ayant un aperçu direct de l'apparence d'un vêtement sans avoir à le retoucher. A un niveau plus ludique il est facile d'imaginer les répercussions que cela aurait. Imaginez-vous dans un monde virtuel et voyez le réalisme que cela apporterait de se voir avec des vêtements ayant un comportement semblable à la réalité ! Il en est de même dans le monde des jeux vidéos et des films réalisés uniquement en synthèse d'images. Cette simulation permettrait également de diminuer considérablement le temps de calcul de chaque image.
Pour atteindre cet objectif plusieurs problèmes majeurs sont à résoudre. Un premier point épineux concerne les calculs dynamiques, c'est-à-dire les calculs du mouvement du tissu au cours du temps. Un second point important concerne la détection de collisions. C'est la partie la plus gourmande en temps de calcul dans une simulation de vêtements. Elle permet notamment l'obtention des plis réalistes du tissu en contact avec une surface.
La méthode que nous avons adoptée pour obtenir du temps réel est la parallélisation des algorithmes, ainsi que l'exécution de l'application sur une grappe de PC. Pour ce faire nous avons utilisé notamment l'environnement de programmation parallèle Athapascan ([BIG], [tea]). Le choix concernant l'utilisation d'une grappe de PC plutôt qu'une machine multi-processeurs dédiée (Cray) est facilement compréhensible. A l'heure actuelle, il est beaucoup moins coûteux de réunir plusieurs ordinateurs ordinaires, que de se doter d'une machine unique multi-processeurs. D'autre part une grappe d'ordinateurs présente une grande souplesse, car il est alors facile de ne changer que quelques processeurs et ainsi de suivre l'évolution des technologies dans ce domaine. Les machines multi-processeurs ne permettent pas cette souplesse, et deviennent donc

[1] Projet INRIA-IMAG APACHE. Travail en collaboration avec F. Faure iMAGIS-GRAVIR. Ce travail a été partiellement financé par le contrat de la thématique prioritaire n°4 *Sciences et technologies de l'information, outils et applications* de la région Rhônes-Alpes.

vite dépassées.
Par la suite, les méthodes employées pour cette simulation dynamique de textiles pourront être étendues dans un cadre plus général aux applications de calculs dynamiques concernant d'autres domaines.
Nous allons donc présenter dans ce papier une parallélisation des algorithmes utilisés dans une simulation de textiles basée sur des modèles physiques. Nous verrons quels modèles physiques nous avons choisis, comment nous avons employé l'environnement de programmation parallèle Athapascan pour la parallélisation de cette application, ainsi que les structures de données utilisées, celles-ci ayant une grande importance dans la parallélisation des algorithmes. Nous détaillerons notamment la méthode de résolution du système linéaire par un gradient conjugué par blocs parallèles. Nous avons commencé par implémenter la partie concernant les calculs dynamiques, plutôt que par la détection de collisons, car ces calculs dynamiques imposent une structure de données précise pour la parallélisation des algorithmes.
Nous verrons les performances que nous avons obtenues par ces méthodes, ainsi que l'intérêt de coupler l'utilisation d'une grappe de PC pour effectuer les calculs, et d'un ordinateur standard pour la visualisation.

2 L'environnement de programmation parallèle Athapascan

L'utilisation de machines parallèles, comme une grappe de PC, permet l'obtention d'une puissance de calcul bien supérieure à celle des machines traditionnelles, sans être aussi coûteuses que la technologie des multi-processeurs. En effet dans les années 70, le parallélisme de matériel correspondait à la construction de processeurs vectoriels (Cray), qui était adapté à un certain type d'applications (structures régulières). Mais ces systèmes très spécifiques étaient trop chers. C'est pourquoi dans les années 90, les machines parallèles ont été élaborées à partir de processeurs puissants standards et d'un réseau d'interconnexion rapide (ces réseaux locaux étant devenus performants). Nous obtenons ainsi une structure de grappe de machines pas trop coûteuse.
Pour pouvoir utiliser facilement les ressources disponibles sur ce type d'architecture parallèle, nous avons employé l'environnement de programmation parallèle Athapascan développé dans le laboratoire ID-IMAG dans le cadre du projet APACHE. Athapascan implémente un modèle de programmation parallèle haut niveau basé sur une mémoire partagée. Les avantages principaux de cet environnement sont : sa facilité de programmation et sa portabilité. Grâce à cela notre application pourra être exécutée sur n'importe quel type de machines parallèles sans à avoir à modifier son code. Cet environnement est composé de deux modules complémentaires :

- Athapascan-0 ([BIG]) est le module exécutif qui permet l'utilisation de la multiprogrammation légère dans un contexte distribué. Ce module, basé sur un couplage entre différentes bibliothèques de processus légers et de communication standards, constitue le noyau exécutif de l'environnement et assure sa portabilité ;
- Athapascan-1 ([tea]) est l'interface applicative au-dessus de Athapascan-0 dont nous nous servons. Il implémente un modèle de programmation haut niveau basé sur un mécanisme de création de tâches collaborant entre elles par l'accès à une mémoire partagée.

Un programme Athapascan-1 consiste en un ensemble de tâches créées dynamiquement pouvant partager des objets. L'exécution d'un programme consiste en l'exécution des tâches à une date déterminée par le système dans le respect des contraintes de précédence dûes aux données partagées. Pour pouvoir employer cet environnement il suffit de découper la simulation en plusieurs tâches de calculs plus ou moins indépendantes, et de déterminer quels seront les objets partagés par les différentes machines qui constituent la grappe de calcul. Les tâches sont alors exécutées dès qu'elles sont prêtes, c'est-à-dire dès que toutes les données dont elles se servent sont disponibles. C'est l'environnement parallèle qui gère l'ordre d'exécution des tâches et les communications entre processeurs, ainsi que les synchronisations. D'autre part il est facile de changer d'ordonnanceur et ainsi de trouver facilement le mieux adapté à notre simulation, c'est-à-dire celui qui nous fournira les meilleures performances.

3 Simulation physique de textiles

Nous décrivons dans cette partie d'une façon générale la modélisation physique des tissus ([BE96], [JL95], [BHG94], [DT88]). La simulation de textiles consiste à simuler le mouvement et la déformation d'un vêtement. Elle entre dans le cadre des simulations dites dynamiques, consistant à simuler le mouvement, la déformation et l'interaction qui peut se produire entre différents objets. Les étapes principales de telles simulations sont les suivantes :

- La représentation du mouvement d'un objet en fonction des forces extérieures et intérieures,
- La détection des collisions entre les différents objets,
- Le calcul de la force de collision entre deux objets.

Pour la modélisation physique du tissu, nous avons repris celle présentée par David Barraf et Andrew Witkin dans leur papier [BW98], détaillant entre autre la méthode d'intégration implicite.

3.1 Modèle physique

Notre simulation de vêtements est basée sur un maillage triangulaire de particules. Nous disposons de n particules dans un espace à 3 dimensions. Sur chaque particule est exercée une force f également à 3 dimensions. Pour conserver les propriétés d'élasticité et de poids du tissu, nous avons choisi une représentation mécanique avec un système de type masse-ressort. Les ressorts étant positionnés entre les particules pour permettre d'exercer des forces sur ces masses.
L'accélération de la i^e particule de notre simulation est $x_i'' = f_i/m_i$, selon la loi fondamentale de la dynamique, où f_i est la force exercée sur cette particule, et m_i sa masse. Si nous définissons la matrice diagonale M par $diag(M) = (m_1, m_1, m_1, ..., m_n, m_n, m_n)$, où $m_1, ..., m_n$ réprésentent les masses des n particules, nous pouvons écrire :

$$x'' = M^{-1} f(x, x').$$

Il ne reste qu'à intégrer cette accélération pour calculer la position et la vitesse des particules. Nous avons pour cela utilisé deux méthodes : Euler explicite et implicite. Pour cette dernière, nous avons un système linéaire creux à résoudre. Nous employons alors la méthode du gradient conjugué exploitant facilement les propriétés des matrices creuses.

3.2 Schéma d'Euler explicite sur le système de particules

Le schéma d'Euler explicite sert à calculer l'évolution de l'état d'un système dans le temps. C'est la plus simple des méthodes numériques. Nous reprenons ici les notations de Andrew Witkin et David Barraf présentées dans le tutorial [WB93]. Soit $x_0 = x(t_0)$ la valeur initiale de x, et $x(t_0 + h)$ l'estimation de x après un temps $t_0 + h$, où h représente le pas de temps. La méthode d'Euler calcule simplement $x(t_0 + h)$ en avançant d'un pas de temps dans la direction de la dérivée,

$$x(t_0 + h) = x_0 + hx'(t_0).$$

Cette méthode peut être instable, dans le cas où h est grand, c'est-à-dire que la solution oscille autour d'une position d'équilibre, voire diverge. Nous avons tout de même implémenté cette méthode afin de la comparer à d'autres et voir ses limites dans le cadre de la simulation de vêtements.

3.3 Schéma d'Euler implicite sur le système de particules

Nous avons vu que les méthodes explicites peuvent diverger dans certains cas, nous utilisons alors des méthodes dites implicites. Nous appliquons ici la méthode d'Euler implicite à notre système de particules. Nous reprenons dans cette partie les notations et formulation données

dans le papier [BW98], et également utilisées dans l'article [VT00a].
Supposons que la position $x(t_0)$ et la vitesse $x'(t_0)$ du système soient données au temps t_0, le but est de déterminer la nouvelle position $x(t_0 + h)$. Pour calculer le nouvel état du système ainsi que sa vitesse en utilisant une méthode implicite, il faut définir la vitesse v comme $v = x'$, et nous obtenons alors :

$$\frac{d}{dt}\begin{pmatrix} x \\ x' \end{pmatrix} = \frac{d}{dt}\begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} v \\ M^{-1}f(x,v)) \end{pmatrix}.$$

Pour simplifier les notations nous définissons $x_0 = x(t_0)$ et $v_0 = v(t_0)$. Nous définissons également $\Delta x = x(t_0 + h) - x(t_0)$, $\Delta v = v(t_0 + h) - v(t_0)$ et la force $f_0 = f(x_0, v_0)$.
David Barraf et Andrew Witkin nous montrent alors ([BW98]) qu'en employant les séries de Taylor, le système précédent devient

$$\left(M - h\frac{\partial f}{\partial v} - h^2\frac{\partial f}{\partial x}\right)\Delta v = h\left(f_0 + h\frac{\partial f}{\partial x}v_0\right), \tag{1}$$

qu'il ne reste plus qu'à résoudre pour obtenir Δv, et ainsi calculer facilement $\Delta x = h(v_0 + \Delta v)$. En résumé, la méthode d'Euler implicite consiste à évaluer f_0, $\frac{\partial f}{\partial x}$ et $\frac{\partial f}{\partial v}$, construire le système d'équations (1), et le résoudre pour obtenir Δv, et enfin mettre à jour x et v.

4 Algorithme du gradient conjugué

Nous cherchons à résoudre le systèmes d'équations (1) afin de trouver les vitesses de nos particules. Définissons la matrice symétrique définie positive **A** par

$$A = \left(M - h\frac{\partial f}{\partial v} - h^2\frac{\partial f}{\partial x}\right), \tag{2}$$

et le vecteur **b** par

$$b = h\left(f_0 + h\frac{\partial f}{\partial x}v_0\right). \tag{3}$$

La matrice A étant symétrique définie positive, nous pouvons appliquer l'algorithme du gradient conjugué pour la résolution du système linéaire $A\Delta v = b$. L'algorithme itère jusqu'à ce que le facteur d'erreur prenne une valeur en dessous de ε qui traduit la précision souhaitée et qui peut être normalisé de façon homogéne selon l'intervalle de temps, les masses des particules et les distances du système. L'algorithme peut s'écrire ([Saa96], [VKK94]) :

Algorithme 1 Algorithme du gradient conjugué pour la résolution de $A\Delta v = b$

```
β ← 0;                          // Initialisation du facteur d'erreur
Δv ← 0;                         // Initialisation de la solution
R ← b − AΔv;                    // Initialisation du vecteur de residu

  α ← R^T R;                    // Initialisation du pas

  Si(β ≠ 0)
    T ← R + (α/β)T;             // Calcul de la nouvelle direction
  Sinon
    T ← R;                      // Initialisation du vecteur de direction

  β ← T^T AT;                   // Calcul du facteur d'erreur
  R ← R − (α/β)AT;              // Calcul du vecteur de residu
  Δv ← Δv + (α/β)T;             // Calcul de la solution
  β ← α;                        // Nouveau facteur d erreur
Jusqu'a (β < ε)                 // Iteration jusqu a la precision souhaitee
```

5 Parallélisation de la simulation de textiles

Nous allons dans cette partie expliciter les calculs des états des particules et des forces qui leur sont appliquées, ainsi que la résolution du système linéaire par une méthode du gradient conjugué par blocs parallèles. Notre algorithme parallèle découpe l'ensemble des particules sur les processeurs. Si les données sont placées dans un tableau, cela revient à découper ce tableau par blocs. Chaque processeur calcule les données seulement pour ses particules. Mais pour calculer les forces appliquées aux particules, les noeuds de calcul ont besoin de connaître la position, la vitesse et l'accélération de toutes les particules voisines. A cette fin nous avons construit une matrice fournissant toutes ces informations, et permettant d'effectuer les calculs par blocs indépendants.
Nous verrons dans un premier temps quelles structures de données nous avons adoptées, puis comment sont calculés les états des particules ainsi que les forces, puis nous verrons une des opérations élémentaires de l'algorithme du gradient conjugué découlant des structures de données (produit matrice/vecteur).

5.1 Parallélisation par blocs de données

Nous avons opté pour une méthode de parallélisation se basant sur un découpage des données, c'est-à-dire que nous effectuons en parallèle des calculs identiques mais sur des données différentes. Dans le cadre de la simulation, ceci consiste à calculer en parallèle les positions, vitesses et accélérations des particules.
Pour pouvoir effectuer indépendemment les calculs des états des particules, nous les avons décomposées en blocs. Les coordonnées en 3 dimensions des positions, vitesses et accélérations des particules à un instant t sont stockées dans 3 tableaux différents, qui sont eux-mêmes découpés en blocs (figure 1).

Figure 1: Structure de données des tableaux des positions, vitesses et accélérations.

Chaque bloc est déclaré comme un objet partagé Athapascan-1, et ainsi nous pouvons effectuer en parallèle les calculs des états sur des blocs de particules différents. Pour mieux comprendre considérons le schéma algorithmique de l'implémentation 2.

Implémentation 2 Schéma général de parallélisation pour des opérations indépendantes

```
struct EulerExplicite
{
  void operator()(Shared_r_w<VectorCoord> BlocVit,
                  Shared_r<VectorCoord> BlocAccel, ...) { ... }
};

for(int i=0; i<Nombre_de_blocs; i++)
   Fork<EulerExplicite>()(Vitesse[i], Acceleration[i], ...);
```

Les positions des particules sont stockées dans le tableau "Position". Position[indice] représente un bloc comprenant la position de N particules, avec N la taille d'un bloc. Ce bloc est un objet partagé au sens de Athapascan-1, et la méthode "Tâche_Calcul_Position" lui est appliquée. Cette fonction va calculer les positions des N particules contenues dans ce bloc. Le mot-clé **Fork** signifie la construction d'une tâche Athapascan-1, et ainsi la boucle "for" permet de lancer "Nombre_de_blocs" tâches en parallèle sur des processeurs différents, calculant chacune les positions des particules contenues dans Position[i]. Nous avons ainsi calculé les positions de toutes les particules par blocs indépendants.
Lors du calcul des forces de chacune des particules, nous sommes amenés à devoir connaître les états des particules voisines. Il n'est donc pas possible dans l'immédiat d'utiliser la même

boucle de calcul que précédemment car pour le calcul des forces des particules contenues dans le bloc i, il faut connaître l'état d'autres particules qui ne sont pas forcément dans le même bloc. Pour remédier à ce problème nous avons construit une matrice permettant de connaître le graphe du maillage, c'est-à-dire de savoir quelles sont les particules voisines d'une particules donnée. Cette matrice est également décomposée en plusieurs blocs partagés, et contient pour une particule donnée les numéros et états de ses voisines (particules reliées à elle par un ressort), les caractéristiques du ressort qui la rattache à une particule donnée, les contributions des forces exercées par cette particule envers ses voisines, ainsi que les coordonnées du vecteur de résidu de la résolution du système linéaire, dans le cadre de la méthode implicite (figure 2).

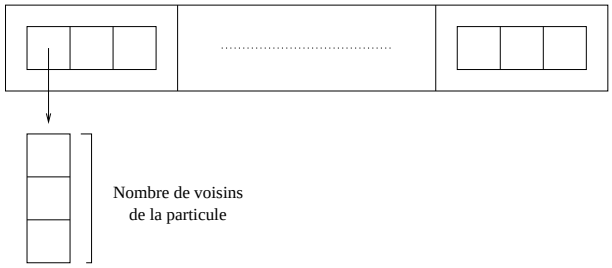

Figure 2: Structure de données de la matrice dite d'adjacence.

Ainsi pour calculer les forces des particules contenues dans un bloc, il suffit de rajouter en paramètre à la tâche le bloc correspondant de la matrice (implémentation 3).

Implémentation 3 Parallélisation du calcul des forces

```
struct Tache_Calcul_Force
{
  void operator()(Shared_r_w<VectorCoord> BlocForce, Shared_r<VectorMatrP>
  BlocMat) { ... }
};

for(int i=0; i<Nombre_de_blocs; i++)
  Fork<Tache_Calcul_Force>()(Force[i], Matrice[i]);
```

Nous obtenons ainsi la possibilité de lancer "Nombre_de_blocs" tâches en parallèle, calculant chacune les forces des particules contenues dans un bloc. Mais cette méthode oblige à dupliquer les informations relatives aux états des particules, car ces derniers sont à la fois stockés dans la matrice et dans les tableaux.
En résumé à chaque pas de temps de la boucle de simulation, nous calculons en parallèle les forces appliquées à un bloc de particules, leur accélération, leur vitesse et leur position. Puis nous devons mettre à jour la matrice avec les nouveaux états des particules.
La taille des blocs est appelé le grain du parallélisme. Pour obtenir une parallélisation de données efficaces, il faut choisir ce grain de telle sorte que les communications dûes à l'envoi des blocs de données sur chacun des processeurs soient moindres que le temps des calculs effectués sur ces blocs, sinon tout le gain obtenu en temps de calcul sera perdu en temps de communication.

5.2 Opération élémentaire du gradient conjugué parallèle

L'algorithme du gradient conjugué est composé d'opérations algébriques de base (combinaison linéaire de vecteurs, produit scalaire, produit matrice/vecteur). L'implémentation 4 présente le produit d'une matrice par un vecteur en 3 dimensions. La matrice et les vecteurs sont découpés en blocs afin de permettre une parallélisation de l'algorithme en créant plusieurs tâches de calculs.
Ce calcul représente le produit $\frac{\partial f}{\partial x}T$ de la résolution du système linéaire, avec T le vecteur de direction et $\frac{\partial f}{\partial x}$ la matrice creuse des contributions des forces. A travers cet exemple, nous voyons que la parallélisation par blocs en utilisant l'environnement parallèle Athapascan est aisée. Chaque tâche de calcul effectue les opérations de calculs sur un bloc des données, et

Implémentation 4 Produit matrice/vecteur $\frac{\partial f}{\partial x}T$

```
void CalculProdMatVect_Task::operator()(Shared_r<VectorMatrP> BlocMat,
Shared_r<VectorCoord> BlocT, Shared_r_w<VectorCoord> BlocW, Shared_r<VectorDer>
BlocContribX, int TailleBloc)
{
   for (int j=0; j<TailleBloc; j++){
    // Calcul des valeurs de df/dx * T de la diagonale
    BlocW.access()[j] = BlocContribX.read()[j] * BlocT.read()[j];

    // Contribution des voisins de la particule j (hors diagonale)
    for(k=0, end_k=BlocMat.read()[j].size(); k<end_k; k++){
      // Calcul des valeurs de df/dx T hors diagonale
      BlocW.access()[j] += BlocMat.read()[j][k].Df_Dx
                         * BlocMat.read()[j][k].Residu;
    }//for_k
   }//for_j
}

void CalculProdMatVect(VectorMatrPP &Mat, VectorCoordP &T, VectorCoordP &W,
VectorDerP &ContribX, int NbBlocs, int TailleBloc)
{
  // Partie du calcul de W = A T de l algorithme du gradient conjugue
  for(int i=0; i<NbBlocs; i++)
   Fork<CalculProdMatVect_Task>()(Mat[i], T[i], W[i], ContribX[i], TailleBloc);
}
```

remplit un bloc du vecteur solution. Mais la matrice de départ $\frac{\partial f}{\partial x}$ n'est pas diagonale, et de ce fait lors du calcul du i^e bloc du vecteur résultat, nous avons besoin de connaître également des valeurs du vecteur T qui se trouvent en dehors du bloc i. Pour remédier à ce problème nous avons entré ces informations dans la matrice d'adjacence (Mat[][][].Residu), et ainsi nous pouvons effectuer le produit par blocs parallèles. Ces valeurs sont mises à jour dès que les valeurs du vecteur T sont modifiées. Dans l'algorithme 4, le tableau ContribX[] contient les éléments diagonaux de la matrice $\frac{\partial f}{\partial x}$ et les éléments non nuls qui ne se trouvent pas sur la diagonale sont stockés dans $Mat[][][].Df\_Dx$. Nous pouvons noter que les données peuvent être partagées en lecture ($Shared\_r <>$) ou en écriture ($Shared\_r\_w <>$). Ces dernières ne peuvent bien entendu n'être accédées que par un processeur à la fois.

5.3 Visualisation de la simulation

Une des particularités de notre simulation de textiles est la combinaison d'une exécution en parallèle des calculs sur une grappe de processeurs, et d'une visualisation sur un ordinateur unique. Nous avons vu comment sont exécutés les calculs en parallèle, leurs résultats étant stockés dans différents tableaux dont notamment celui contenant les positions en 3 dimensions des n particules. L'affichage de ces n particules se fait à l'aide de la librairie GLUT (OpenGL Utility Toolkit). Pour pouvoir coupler son utilisation avec celui de l'environnement parallèle Athapascan, nous avons créé deux programmes séparés : un programme Athapascan-1 (implémentation 6) qui calcule les états des particules, et un programme GLUT (implémentation 5) qui affiche ces états. Nous n'avons pas pu faire un seul et unique programme car les fonctions GLUT n'autorisent aucun paramètre, or les variables partagées Athapascan-1 dans le cadre distribué ne peuvent être déclarées en variables globales, et donc les opérations de calcul ne peuvent être intégrées dans la fonction "update()" et ainsi faire partie de la boucle GLUT. La liaison entre les deux programmes se fait à l'aide d'une variable globale commune contenant les positions des particules.

Implémentation 5 Visualisation des etats des particules

```
// Variable globale contenant les positions de toutes les particules
extern VectorCoordP PosAffichees;

void display(void) { ... Affiche(PosAffichees); ... }
void update() { ... glutPostRedisplay(); ... }

int main () {
  glutDisplayFunc(display); // Affichage
  glutIdleFunc(update);     // Mise a jour de la position des particules
  glutMainLoop();           // Boucle principal
}
```

Implémentation 6 Calcul des etats des particules

```
// Variable globale contenant les positions de toutes les particules
VectorCoordP PosAffichees;

// Mise a jour de la variable globale a partir du tableau des positions
void MiseAjour(VectorCoordP PosAffichees, VectorCoordP PosCalculees, int
NbBlocs) {
   for (int i=0; i<NbBlocs; i++)
    Fork<Tache_MiseAjour>()(PosCalculees[i], PosAffichees[i]);
}

int main ()
{
   CalculForce(Force, Matrice, ...);
   CalculAcceleration(Acceleration, ...);
   CalculVitesse(Vitesse, ...);          // Calcul par integration implicite
   CalculPosition(PosCalculees, ...);
   CalculMatrice(Matrice, ...);          // Mise a jour de la matrice
   MiseAJour(PosCalculees, PosAffichees, NbBlocs);
}
```

6 Résultats

Afin d'évaluer les performances de notre simulation de textiles, nous avons fait varier le nombre de particules et son mode d'exécution (SMP, distribué). En effet l'environnement parallèle Athapascan permet ces deux modes d'exécution sans à avoir à modifier le code de l'application. Ces tests sont effectués aussi bien sur la version explicite qu'implicite de l'intégration au cours du temps. Nous avons pour cela fixé le nombre d'itérations de la boucle de simulation à 3 et celui du gradient conjugué pour la méthode implicite à 10. De plus nous verrons le temps passé dans chaque partie des algorithmes (calculs des accélérations, positions, vitesses, et du gradient conjugué pour la méthode implicite). Ces tests ont été réalisés sur une grappe de 216 HP e-vectra (pentium III, 733 MHz, 256 Mo, 15 Go) déployée dans le cadre du projet i-cluster (INRIA, HP) et du projet LIPS (INRIA, BULL) pour l'exécution en distribué, et sur une machine quadri-processeurs (Pentium Pro, 200 MHz, 256 Mo) pour l'exécution en SMP.
Le temps relatif au calcul des vitesses dans le cadre implicite ne tient pas compte du temps de la mise à jour de la matrice du maillage (figures 6 et 4). En effet nous pouvons observer que ce temps est relativement long. Ceci s'explique par le fait que l'implémentation actuelle n'est pas optimisée. Elle implique la construction de nombreuses petites tâches indépendantes qui sont trop coûteuses en mode distribuée.
La version distribuée ne donne pas d'énormes performances (figure 6) en comparaison à l'exécution sur un seul processeur. Mais pour ces temps nous avons utilisé une version de l'environnement

Athapascan qui n'était pas encore optimale, la nouvelle version étant en cours de modification.

Cas implicite : version distribuée sur 10 processeurs

Nombre de particules	Temps/Frame (réel s)	Temps de chaque fonction en ms				
		Force	Accel	Vitesse	Position	MatAdj
2 500	2.70/2.75	0.31/0.298	0.133/0.104	18.42/11.402	0.181/0.164	437/373
10 000	11.46/11.9	0.789/0.211	0.327/0.08	49/81	0.441/0.097	215/178
22 500	32/35.3	1.81/0.86	0.741/0.346	212/250	26.8/57.5	706/609

Cas explicite : version distribuée sur 10 processeurs

Nombre de particules	Temps/Frame (réel s)	Temps de chaque fonction en ms				
		Force	Accel	Vitesse	Position	MatAdj
2 500	0.35/0.34	0.236/0.151	0.128/0.071	0.128/0.071	0.159/0.103	358/339
10 000	1.62/2.02	1.43/0.159	0.313/0.072	0.325/0.071	0.402/0.09	162/202
22 500	3.66/4.45	1.17/0.198	0.702/0.103	0.713/0.104	0.906/0.126	366/445

Figure 3: Performances de la simulation de textiles en distribuée. Temps/Frame indique le temps réel moyen pour chaque frame. Le temps est donné également pour 5 tâches de calcul (force, accélération, vitesse, position et mise à jour de la matrice). La première valeur indiquée correspond à une exécution sur 1 processeur, et l'autre à une exécution sur 10 processeurs.

Cas implicite : version SMP sur 4 threads

Nombre de particules	Temps/Frame (réel s)	Temps de chaque fonction en ms				
		Force	Accel	Vitesse	Position	MatAdj
90 000	49.6/49.6	0.148/0.011	0.056/0.041	156/334	0.05/0.05	12 786/12 596
122 500	92/100	2.4/0.13	0.07/0.052	270/711	0.059/0.058	17 367/18 920
160 000	125/221	383/165	322/51.2	112/465	0.072/0.056	22 000/111 105

Cas explicite : version SMP sur 4 threads

Nombre de particules	Temps/Frame (réel s)	Temps de chaque fonction en ms				
		Force	Accel	Vitesse	Position	MatAdj
90 000	1.08/1.11	0.117/0.081	0.048/0.116	0.03/0.019	0.045/0.035	1 081/1 110
122 500	13/1.51	0.146/0.077	0.057/0.063	0.037/0.019	0.053/0.148	13 276/1 513

Figure 4: Performances de la simulation de textiles en SMP. Temps/Frame indique le temps réel moyen pour chaque frame. Le temps est donné également pour 5 tâches de calcul (force, accélération, vitesse, position et mise à jour de la matrice). La première valeur indiquée correspond à une exécution sur 1 thread, et l'autre à une exécution sur 4 threads.

7 Perspectives

- **Amélioration de la parallélisation actuelle :** Le découpage par blocs indépendants des structures de données, permet la réalisation des calculs en parallèle sur chaque processeur. Mais pour que chaque processeur puisse effectuer les calculs relatifs à son bloc de données, il a besoin de données d'un autre bloc. Nous sommes alors obligé de dupliquer certaines informations pour permettre les calculs en parallèle. Cette duplication est réalisée dans la matrice du graphe du maillage.
  Une amélioration consisterait en la renumérotation des particules du tissu, dans le but de minimiser ce nombre d'informations dupliquées. En effet cette redondance apparaît lors du calcul des forces exercées sur une particule (la connaissance des caractéristiques des particules voisines est alors nécessaire) et lors de la résolution du système linéaire dans le cadre de la méthode implicite. La renumérotation consisterait alors à essayer de mettre

dans un même bloc de données des particules voisines, c'est-à-dire à faire attention à leur localisation dans l'espace de simulation.

- **Détection et traitement des collisions [VT00b] [MMP00] :** Notre simulation de textiles ne traite pas pour le moment de la détection des collisions entre particules. Celle-ci sera prochainement intégrée à notre application.

Références

[BE96] W. Stasser B. Eberhardt, A. Weber. A fast, flexible, particle-system model for cloth. *IEE Computer Graphics and Applications*, 16:52–59, 1996.

[BHG94] D. Breen, D. House, and P. Getto. A particle-based model for simulating the draping behavior of woven cloth. *Textile Research Journal, Vol. 64*, 11:663–685, nov 1994.

[BIG] J. Briat and M. Pasin I. Ginzburg. *Athapascan-0 User Manual.* Projet APACHE, Grenoble.

[BW98] D. Barraff and A. Witkin. Large steps in cloth simulation. In *Computer Graphics Proceedings, Annual Conference Series*, pages 43–54. SIGGRAPH, 1998.

[DT88] M. Kass D. Terzopoulos, A. Witkin. *Computer graphics techniques for modeling cloth.* 1988.

[JL95] D. Crochemore J. Louchet, X. Provot. Evolutionary identification of cloth animation models. In Dimitri Terzopoulos and Daniel Thalmann, editors, *Computer Animation and Simulation'95*, pages 44–54. Springer-Verlag, 1995.

[MMP00] A. Mir M. Mascaro and F. Perales. Elastic deformations using finite element methods in computer graphic publications. In H. H. Nagel and F. J. Perales, editors, *LNCS*, volume 1899, pages 38–47, 2000.

[Saa96] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, 1996.

[tea] Athapascan-1 team. *Athapascan-1.* Projet APACHE, Grenoble.

[VKK94] A. Gupta V. Kumar, A. Grama and G. Karypis. *Introduction to parallel computing, design and analysis of algorithms.* Benjamin/Cummings, 1994.

[VT00a] P. Volino and N. Magnenat Thalmann. Accurate collision response on polygonal meshes. In *CA'00 Computer Animation 2000*, Geneva, June 2000.

[VT00b] P. Volino and N. Magnenat Thalmann. Implementing fast cloth simulation with collision response. In *CGI'00 Computer Graphics International*, Geneva, June 2000.

[WB93] A. Witkin and D. Baraff. Differential equations basics. In *SIGGRAPH93 20th International Conference on Computer Graphics and Interactive Techniques*, pages B1–B8, California, August 1993.