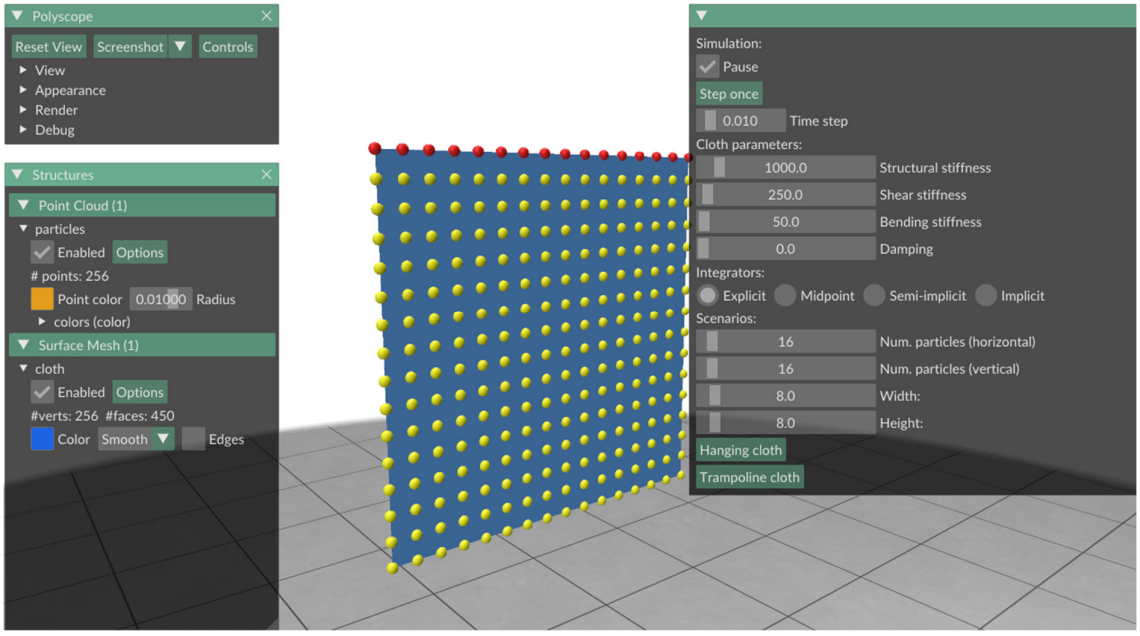
Devoir #1

Simulation de tissu



Rédigé par Sheldon Andrews
Version 1.5 (14 mai 2025)

Introduction

La simulation de tissu est l'un des types de simulation les plus courants dans les jeux vidéo. Il est souvent utilisé pour ajouter plus de réalisme aux mouvements des personnages pour l'accentuer avec des effets secondaires. Dans ce devoir, vous allez implémenter une simulation de tissu, en plus de diverses techniques d'intégration numérique. Les endroits où vous devez implémenter du code sont indiqués par un commentaire `// TODO` dans le code de départ.

Code de départ et dépendances

Compilateur C++. Vous devrez installer un compilateur C++. Cela dépend de votre plate-forme et de votre système d'exploitation. Sous Windows, je recommande *Visual Studio 2019* ou *2022*, que vous pouvez télécharger et installer gratuitement. Sous Linux, *gcc 7.x* est supporté. Sous macOS, *Xcode 11* est supporté. Vous pouvez trouver plus d'informations sur des compilateurs et des plateformes spécifiques dans les liens ci-dessous.

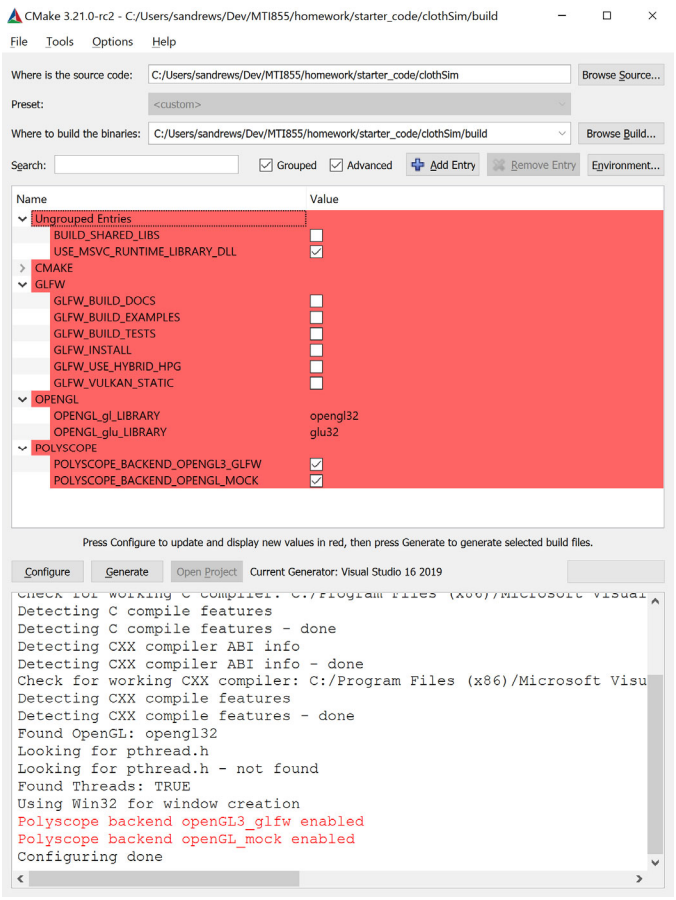
Eigen. Une copie de la bibliothèque d'algèbre linéaire Eigen (http://eigen.tuxfamily.org/index.php) est fournie avec le projet. Plusieurs des structures de données du code utilisent Eigen pour stocker les matrices et les vecteurs. Un guide sur l'utilisation d'Eigen est disponible ici:
https://eigen.tuxfamily.org/dox/GettingStarted.html
… et voici le guide de référence rapide :
https://eigen.tuxfamily.org/dox/group__QuickRefPage.html

Polyscope. Une copie de Polyscope (https://polyscope.run/) est également incluse, qui est utilisé pour visualiser et pour rendre les simulations. Note : Polyscope utilise ImGUI pour l'interface d'utilisateur. Vous êtes libre (et encouragé) de créer votre propre interface pour tester votre code.

CMake. Les fichiers de projet pour compiler le code en C++ sont produit par CMake. Si vous travaillez sous *Windows*, avec *Visual Studio* :

1. Exécuter l'interface graphique de CMake.
2. Mettre le chemin du code source au dossier `MTI855-Devoir01` et mettre le chemin pour compiler les binaries au dossier `MTI855-Devoir01/build`
3. Appuyer sur le bouton *Configurer* et spécifier le compilateur et la plateforme du projet (par ex. *Visual Studio 17 2022* et *x64*)
4. Appuyer sur le bouton *Générer* pour créer les fichiers du projet.
5. Dans le dossier `MTI855-Devoir01/build`, identifier le fichier `MTI855-devoir01.sln` et l'ouvrir avec *Visual Studio 2022*.
6. Dans Visual Studio, faire CTRL-B pour compiler et F5 pour exécuter le programme.

Par exemple, voici une capture d'écran de la configuration du projet CMake à l'aide de l'interface graphique :



Si vous travaillez avec un terminal et l'outil *CMake* (sous Linux) :

1. Ouvrir un terminal et se rendre dans le dossier `MTI855-Devoir01`.
2. Entrer la commande suivante :
   `$ mkdir build && cd build && cmake ..`
3. Ensuite, faire make pour compiler et *./tissu* pour exécuter le programme

Instructions

Interaction avec la souris

Bouton scroll: mode zoom
Bouton gauche : mode de rotation
Bouton droit : mode de translation dans le plan de vue

Utilisez **Ctrl + bouton gauche** de la souris pour appliquer une force de ressort qui déplace la particule sélectionnée vers le curseur, et **Ctrl + bouton droit** de la souris pour fixer / détacher la particule sélectionnée.

Objectifs

Assembler le vecteur d'état `q`

Implémentez la fonction `getState(Eigen::VectorXf& q)` dans `ParticleSystem.cpp`. Cela nécessite que vous parcouriez toutes les particules de `m_particles` et que vous mettiez les positions et les vitesses de chacune au bon endroit dans le vecteur `q`.

Le vecteur `q` a la dimension *6n* et la même structure de mémoire comme que celle montrée dans la figure 1. Il y a six (6) valeurs pour chaque particule- une position et une vélocité 3D.

La fonction `setState(const Eigen::VectorXf& q)` a déjà été implémentée pour vous, et il effectue l'opération inverse, ex. mise à jour des positions et vélocités des particules du vecteur `q`. Veuillez l'utiliser comme guide pour écrire la fonction `getState` .

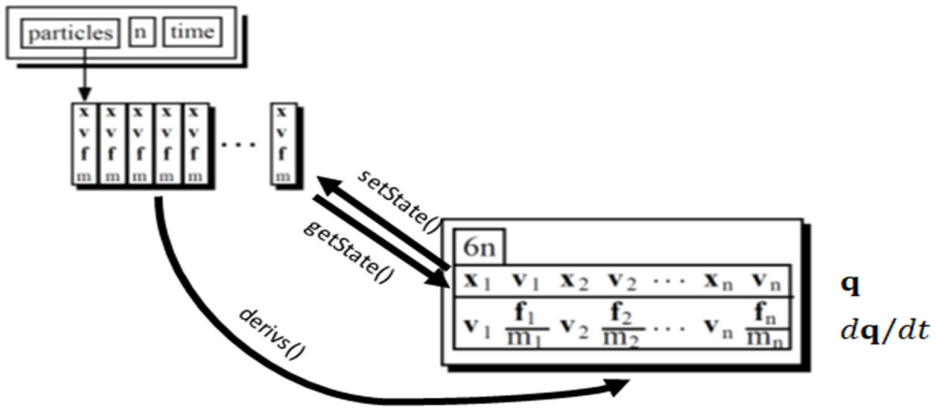


Fig. 1: La configuration en mémoire des vecteurs $\mathbf{q}$ et $d\mathbf{q}/dt$, et les fonctions de la classe `ParticleSystem` pour les obtenir et modifier.

Calcul des forces

Accumulez toutes les forces agissant sur chaque particule dans `ParticleSystem::computeForces()` qui comprend: i) la gravité, ii) les forces élastiques du ressort (loi de Hooke) et iii) les forces d'amortissement du ressort. Après le retour de cette fonction, chaque particule doit avoir les forces accumulées stockées dans la variable membre `Particle::f` qui est un vecteur 3D.

Rédigé par Sheldon Andrews
Version 1.5 (14 mai 2025)

Calcul de la dérivée `dq/dt`

Calculez la dérivée première de l'état du système de particules dans la fonction `ParticleSystem::derivs(Eigen::VectorXf& dqdt)`. Il y a six (6) valeurs correspondant à chaque particule-- une vélocité et une accélération 3D.

Notez: les particules dont la variable membre `Particle::fixed` est `true` doivent avoir un `dq/dt = 0` car elles ne peuvent pas se déplacer.

Implémentation d'intégrateurs numériques

Implémentez les méthodes Euler explicite, midpoint et Euler semi-implicite dans les fichiers `ExplicitEuler.hpp`, `Midpoint.hpp` et `SemiImplicitEuler.hpp` respectivement.

Il faut implémenter une fonction pour chacune -- `step(ParticleSystem* particleSystem, float dt)` -- qui avance le système de particules d'un pas de temps dt. Utilisez les fonctions `getState`, `setState` et `derivs` pour assembler le vecteur d'espace de phase et modifier l'état de simulation.

Assembler les matrices de masse et *df*/*dx*

Construisez la matrice de df/dx pour les système de particules en utilisant la fonction `ParticleSystem::dfdx`. Chaque matrice a une dimension de 3×3. Vous devrez boucler sur les ressorts dans le system et calculer les matrices par ressort.

Implémentation d'Euler implicite

Pour implémenter la méthode d'Euler implicite, vous devrez résoudre un système linéaire composé de masse, de rigidité et d'amortissement :

```
A = M - dt*dt*dfdx
```

… et le vecteur de droite :

```
b = dt*f0 + dt*dt*dfdx*v0
```

Cependant, la matrice globale ne sera pas assemblée. Au lieu, une version « sans matrice » de la méthode préconditionnée du gradient conjugué sera utilisée pour résoudre les mises à jour des vélocités.

Mettez l'implémentation de l'intégrateur Euler implicite ans le fichier `ImplicitEuler.hpp`, et votre implémentation du solveur de gradient conjugue dans le fichier `MatrixFreePGS.cpp`.

Note : Pour les particules où `fixed == true`, il faut mettre la solution de `deltav` a zéro.

Évaluation

Grille d'évaluation

Évaluation	Pondération
Implémentez `ParticleSystem::getState` qui assemble le vecteur d'état `q` du système de particules	10
Implémentez `ParticleSystem::derivs` qui calcule le `dqdt` du vecteur d'état `q`	10
Calculez les forces (gravité, élasticité, *damping*)	15
Implémentez les intégrateurs numériques : Euler explicite, midpoint et Euler semi-implicite	15
Calculez la matrice de rigidité dans la fonction `ParticleSystem::dfdx`	15
Implémentez Euler implicite dans `ImplicitEuler.hpp`	10
Implémentez le solveur PGS « sans matrice » dans `MatrixFreePGS.cpp`	20
Style de programmation (commentaires, clarté du code, etc.)	5
Total	**100**