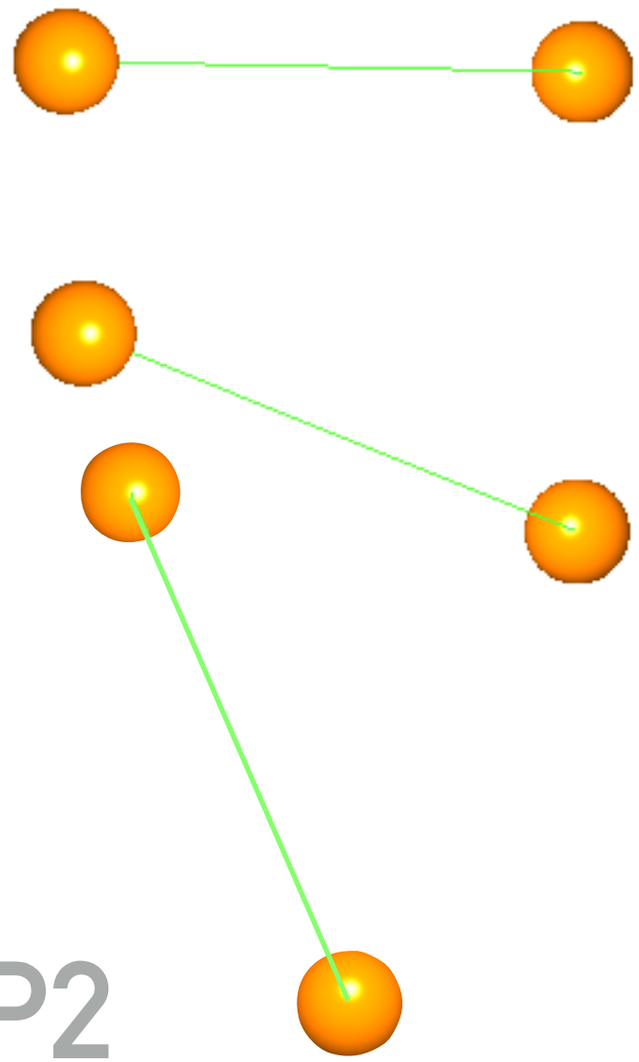
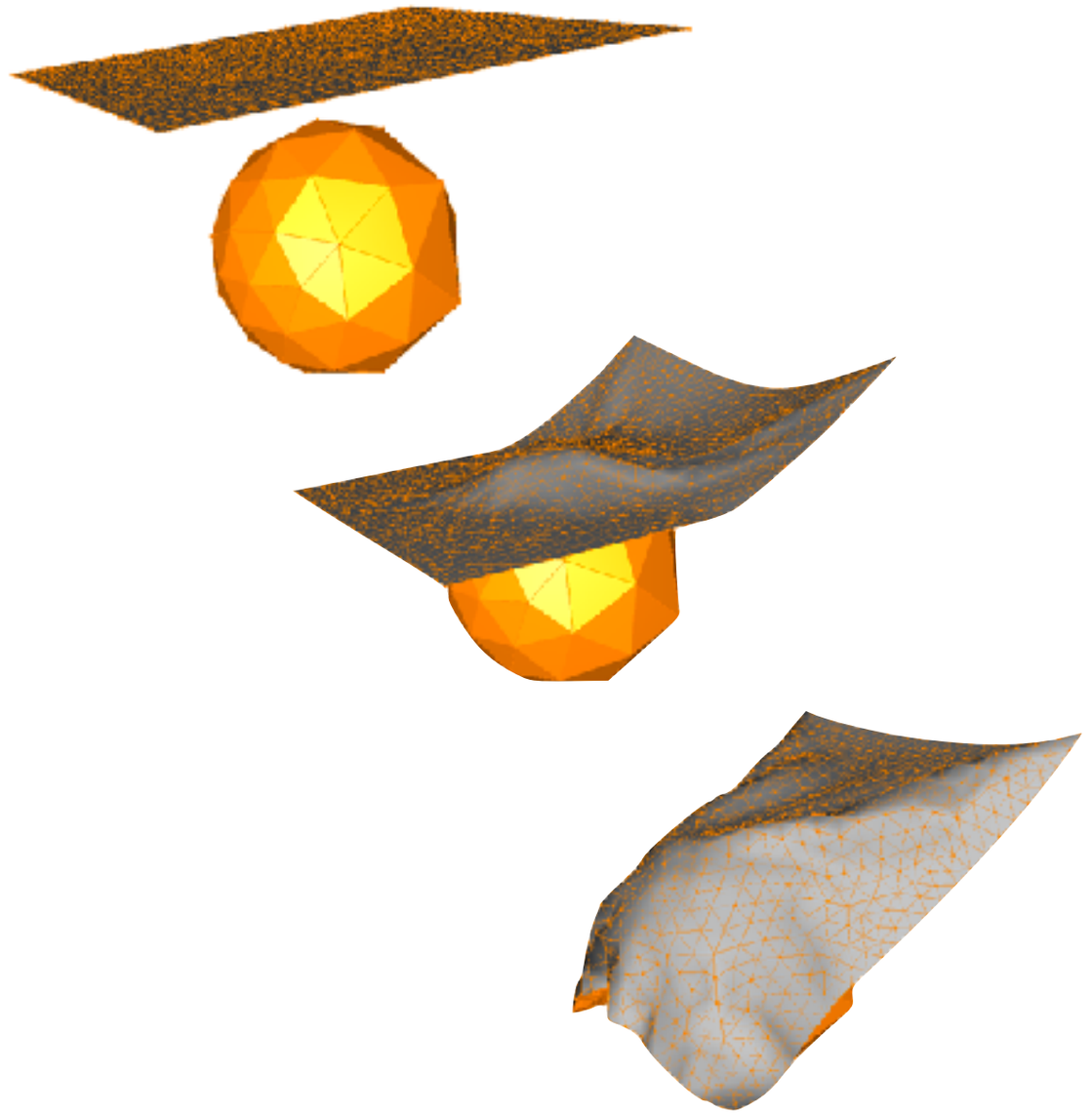




TP2

D'UN PENDULE IMPLICITE A LA SIMULATION D'UN DRAP

ETAPE 1 : CREATION DU PENDULE IMPLICITE DANS SOFA

- Ajouts des solveurs à la place du PythonScriptController

```
springs= rootNode.createChild('Springs')
springs.createObject('EulerImplicitSolver', firstOrder="0", rayleighStiffness=0.01, rayleighMass='0')
springs.createObject('SparseLDLSolver', name='solver', template='CompressedRowSparseMatrixd')
springs.createObject('GenericConstraintCorrection', solverName='solver')
```

- On garde le mechanical object et le maillage

```
springs.createObject('MechanicalObject', position='0 0 0  10 0 0  15 0 10 ', template="Vec3d", showObject="1", name="mObject")
springs.createObject('Mesh', position='@mObject.position', edges ='0 1 1 2 ', name='Mesh', drawEdges='1')
```

- On défini le calcul des forces et contraintes (point fixe)

```
springs.createObject('MeshSpringForceField', linesStiffness='15')
springs.createObject('UniformMass', totalMass='0.3')
springs.createObject('FixedConstraint', indices='0 ')
```

CREATION D'UN SYSTÈME MATRICIEL

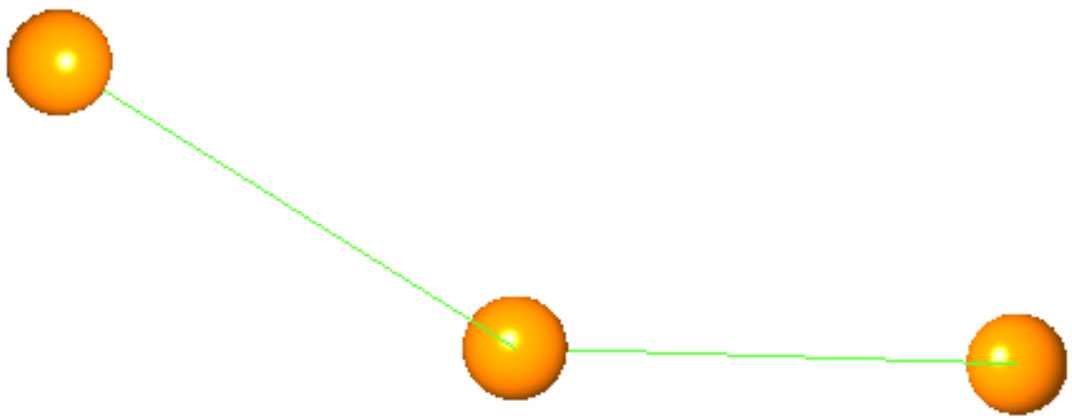
- Beaucoup plus de calcul… mais fait en C++
- Calcul des dérivées des forces internes

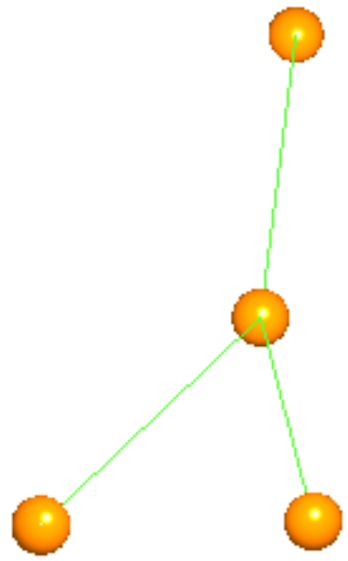
$$\underbrace{\left(\mathbf{M} + h\frac{\delta \mathbb{F}}{\delta \mathbf{v}} + h^2 \frac{\delta \mathbb{F}}{\delta \mathbf{q}}\right)}_{\mathbf{A}} \underbrace{d\mathbf{v}}_{\mathbf{x}} = \underbrace{-h^2 \frac{\delta \mathbb{F}}{\delta \mathbf{q}} \mathbf{v}_i - h\left(\mathbf{f}_i + \mathbf{p}_f\right)}_{\mathbf{b}}$$

- Résolution du système matriciel

A VOUS DE JOUER...

- Simulation du pendule en implicite
  - Questions:
    - Testez différents pas de temps, que constatez-vous ?
    - Testez différentes valeurs de masse et raideur, que constatez vous ?
    - Modifier la topologie du maillage pour faire des pendules plus complexes...





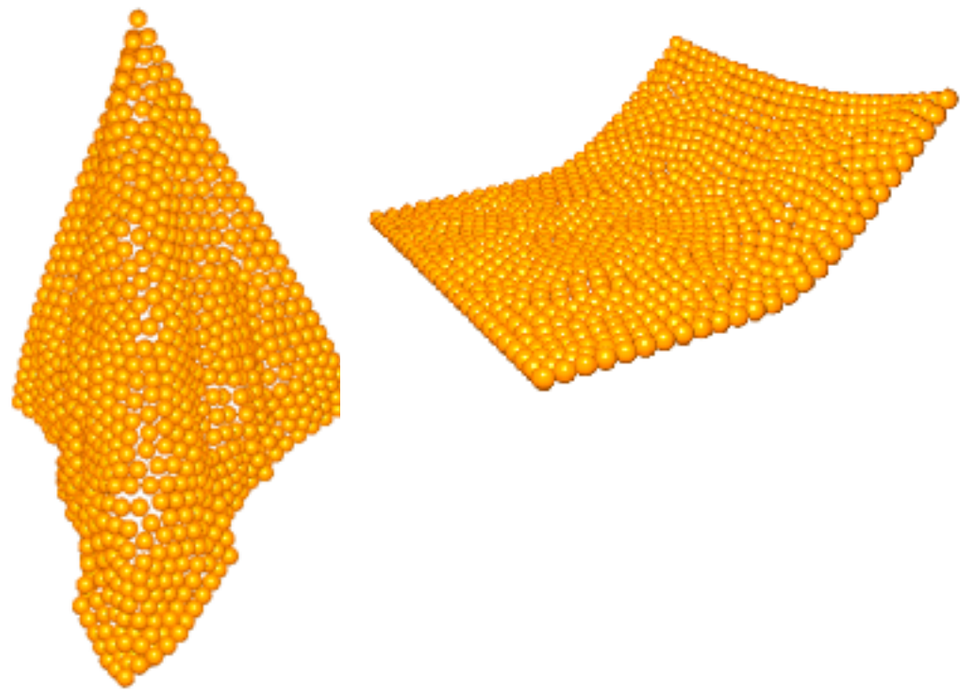
ETAPE 2 : MODÉLISATION (SIMPLE) D'UN DRAP

- Chargement d'un maillage avec le modèle du drap

```
springs.createObject('MeshObjLoader', filename='mesh/square_xz.obj', name='meshOBJ', scale='0.1')
springs.createObject('Mesh', src='@meshOBJ', name='Mesh', drawEdges='1')
springs.createObject('MechanicalObject', template="Vec3d", showObject="1", name="mObject")
```

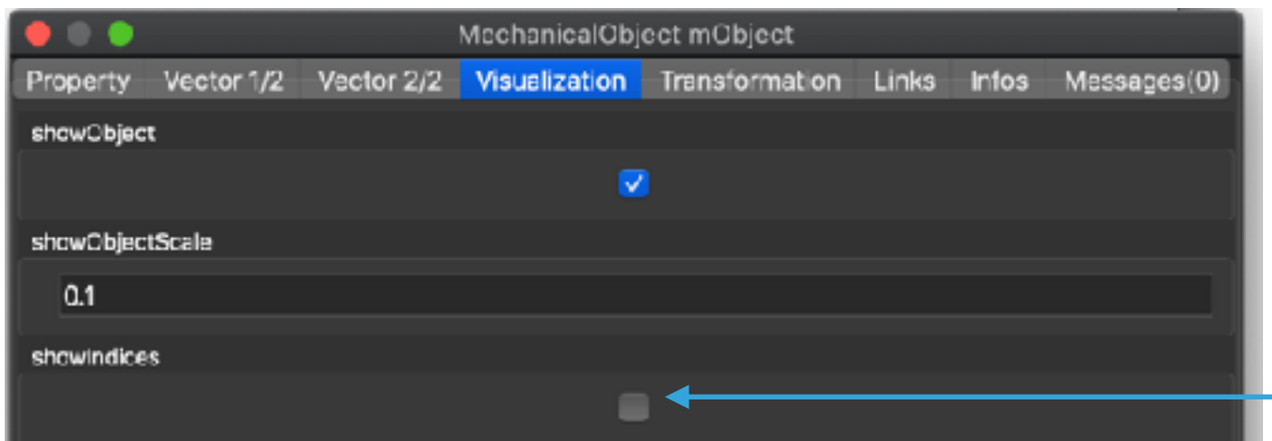
- On change le rayon des sphères pour la visu

```
visualNode= springs.createChild('Visual')
visualNode.createObject('Sphere', radius='0.1')
```



- On peut changer les contraintes

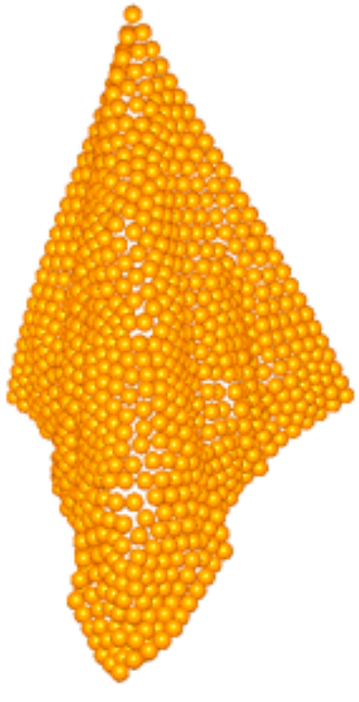
```
springs.createObject('FixedConstraint', indices='0 213')
```

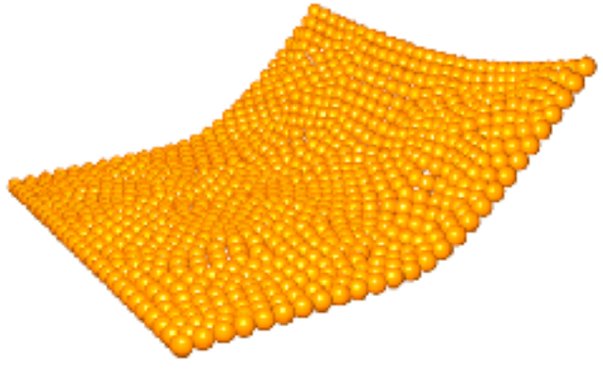


Pour voir les indices

A VOUS DE JOUER...

- Simulation du drap en implicite
  - Questions:
    - Testez différents pas de temps, que constatez-vous ?
    - Testez différentes valeurs de masse et raideur, que constatez vous ?
    - Pouvez-vous expliquer certaines limitations ?





ETAPE 3 : MODÉLISATION DES CONTACTS

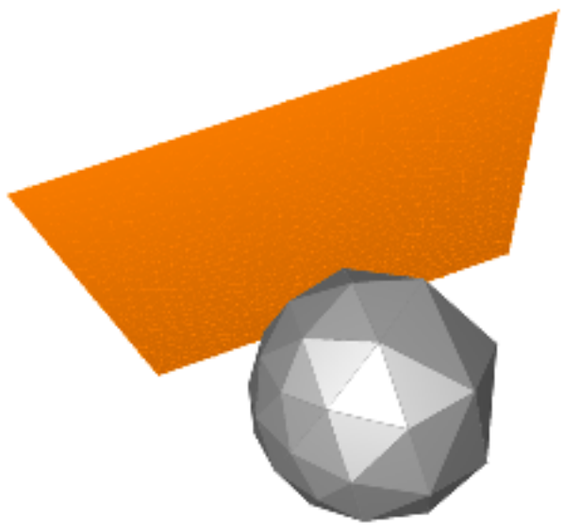
- Ajout du modèle de collision sur le drap

```
springs.createObject('Triangle')
springs.createObject('Line')
springs.createObject('Point')
```

- Définition d'un modèle pour l'obstacle

```
obstacle = rootNode.createChild('Obstacle')
obstacle.createObject('MeshObjLoader', filename='mesh/sphere_02b.obj', name='meshOBJ', scale='0.03',  translation='0 -2 0')
obstacle.createObject('Mesh', src='@meshOBJ', name='Mesh', drawEdges='1')
obstacle.createObject('MechanicalObject')

obstacle.createObject('Triangle', moving='0', simulated='0')
obstacle.createObject('Line',  moving='0', simulated='0')
obstacle.createObject('Point',moving='0', simulated='0')
```



ETAPE 3 : MODÉLISATION DES CONTACTS

- Ajout des solveur de contraintes

```
rootNode.createObject('FreeMotionAnimationLoop')
rootNode.createObject('GenericConstraintSolver', tolerance='1e-7', maxIterations='200')
```

- Pipeline de collision

```
rootNode.createObject('CollisionPipeline', verbose="0");
rootNode.createObject('BruteForceDetection', name="N2");
rootNode.createObject('CollisionResponse', response="default");
rootNode.createObject('LocalMinDistance', name="Proximity", alarmDistance="0.15", contactDistance="0.015");
```

- Choix de la « réponse »

```
rootNode.createObject('RuleBasedContactManager', rules='0 * FrictionContact?mu='+str(0.0),
                                                 name='Response', response='FrictionContact')
#rootNode.createObject('CollisionResponse', response="default");
```

PAR DEFAULT = PENALITE ! (ressorts)

A VOUS DE JOUER...

- Simulation du contact
  - Questions:
    - Testez la solution en contrainte / en pénalité
    - Pour la pénalité, baisser le pas de temps jusqu'à obtenir une simulation stable
    - Quelles sont les différences de comportement ?