Commented Examples

March 12, 2007

This document refers to examples located in directory `sofa/scenes/commentedExamples`.

Contents

1 An oscillating particle

Figure 1 shows the mass-spring system we simulate.

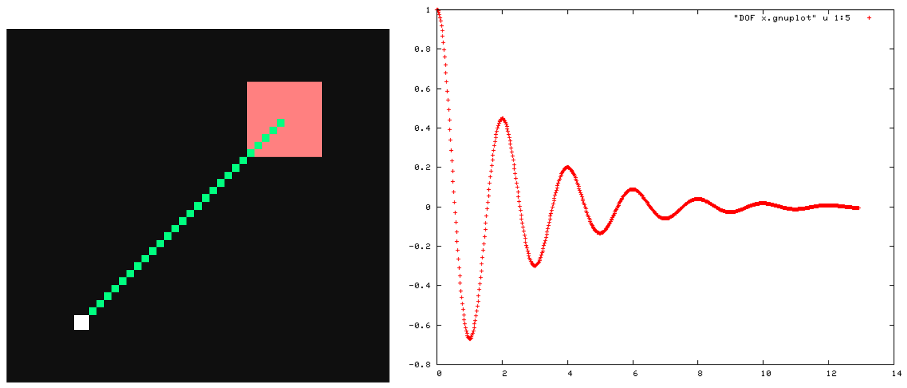


Figure 1: Left: white: moving particle, red: fixed particle, green: spring; Right: an x(t) trajectory.

This scene is modeled for Sofa in `massSpring.scn` as:

```
<Node name="root" dt="0.02" showBehaviorModels="1" showForceFields="1">
   <Node name="mass-spring system">
```

```
        <Object type="MechanicalObject" name="DOF" position="0 0 0  1 0 0"
            velocity="0 0 0  0 1 0"/>
        <Object type="FixedConstraint" name="FixedConstraint" indices="0" />
        <Object type="UniformMass" name="totalMass" mass="1"/>
        <Object type="StiffSpringForceField" name="Spring" spring="0 1 10. 0. 1."
            />
        <Object type="Gravity" gravity="0.0 0.0 0.0"/>
        <Object type="Euler" symplectic="0"/>
    </Node>
</Node>
```

The scene is structured using *nodes* and *objects.* Two nodes are used. The first, called `root`, represents the set of all simulation objects, which we call the *scene.* The other node is called `mass-spring system`. It models our system.

A *system* corresponds to a node. It is made of *components*, called objects in the xml files, and sub-systems corresponding with children nodes.

Our system is described using six components, respectively:

- *MechanicalObject* represents the degrees of freedom and their associated vectors. We have two particles at $0, 0, 0$ and $1, 0, 0$. The default velocities are null.
- *FixedConstraint* says that particle with index 0 is fixed.
- *UniformMass* stores one single mass value for all the particles.
- *StiffSpringForceField* describes a list of springs, each of them modeled using index pair, stiffness, damping and rest length.
- *Gravity* says that the gravity is null.
- *Euler* says that the standard explicit Euler integration method should be applied.

Notice that the scene description does not only describe the objects to simulate, but also the algorithms to apply.

This model is convenient because we can easily try variants. We can of course modify the numerical parameters. More importantly, we can replace one arbitrary component by another of the same kind, all the others remaining unchanged. For example, we can replace *Euler* by *CGImplicit* to compare explicit and implicit time integration and to draw curves using the gnuplot export option.

Each node and object described in the xml file has an associated class in the Sofa library. All the systems are implemented by the single `GNode` class, while the components are implemented using class hierarchies. The *MechanicalObject* and the components which directly manipulate it are templated by `DataTypes` for code genericity.

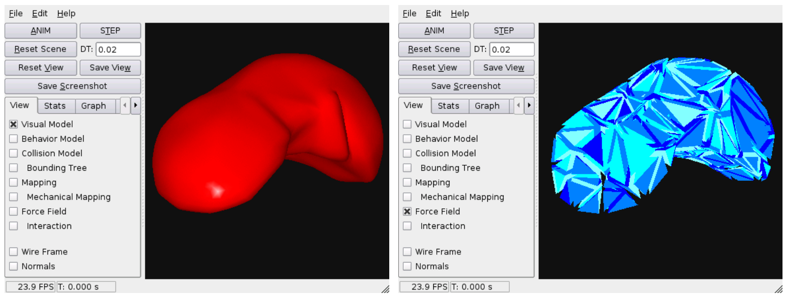


Figure 2: Liver with tetrahedral FEM. Left: view of the visual model. Right: view of the force fields.

2 Livers

Figure 2 shows the liver we simulate.

The model is described in file `liverSpring.scn` as:

```
<Node name="root" dt="0.02" showBehaviorModels="0" showCollisionModels="0"
    showMappings="0" showForceFields="0">
    <Node name="Liver">
        <Object type="CGImplicit" iterations="25"/>
        <Object type="MechanicalObject" template="Vec3f" name="Liver"
            filename="BehaviorModels/liver.xs3"/>
        <Object type="DiagonalMass" name="mass"
            filename="BehaviorModels/liver.xs3"/>
        <Object type="Mesh" name="meshTopology" filename="Topology/liver.mesh"/>
        <Object type="TetrahedronFEMForceField" name="FEM" youngModulus="500"
            poissonRatio="0.3" computeGlobalMatrix="false" method="large"/>
                <Object type="FixedConstraint" name="FixedConstraint" indices="3
                    39 64" />
        <Node name="Visu">
            <Object type="OglModel" name="VisualModel"
                filename="VisualModels/liver-smooth.obj" color="red" />
            <Object type="BarycentricMapping" object1="../.."
                object2="VisualModel" />
        </Node>
    </Node>
</Node>
```

The model has a *Topology* and a *TetrahedronFEMForceField*. The first describes a topology a lists of elements such as edges, triangles, quads, tetrahedra. It is built on data given in auxiliary file.

The second uses the list of tetrahedra of the topology to apply tetrahedron-based FEM forces.

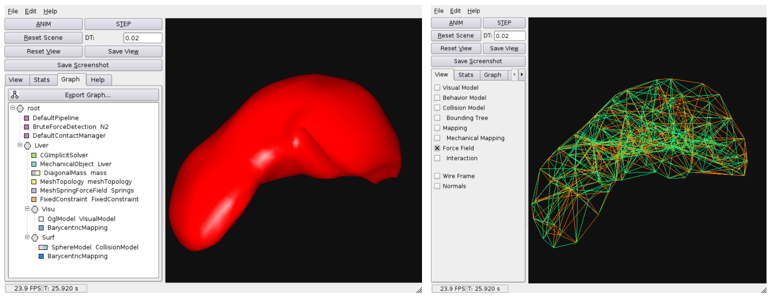
If desired, we can easily replace the TetrahedronFEMForceField with a set of springs based on the edges of the same tetrahedra. We simply replace it by the following line, from file `liverSpring.scn`:

```
<Object type="MeshSpringForceField" name="Springs" tetrasStiffness="400"
    tetrasDamping="4"/>
```

The resulting model is shown in figure 3.

3 An ffd-based liver model

Figure 4 shows the model we simulate. The red surface is a visual model



DT: 0.02

```
root
  DefaultPipeline
  BruteForceDetection  N2
  DefaultContactManager
  Liver
    CGImplicitSolver
    MechanicalObject  Liver
    DiagonalMass  mass
    MeshTopology  meshTopology
    MeshSpringForceField  Springs
    FixedConstraint  FixedConstraint
    Visu
      OglModel  VisualModel
      BarycentricMapping
    Surf
      SphereModel  CollisionModel
      BarycentricMapping
```

23.9 FPS T: 25.920 s

- [ ] Visual Model
- [ ] Behavior Model
- [ ] Collision Model
- [ ] Bounding Tree
- [ ] Mapping
- [ ] Mechanical Mapping
- [✓] Force Field
- [ ] Interaction
- [ ] Wire Frame
- [ ] Normals

Figure 3: The same model with tetrahedral FEM replaced by springs. Left: the scene graph. Right: the force field.

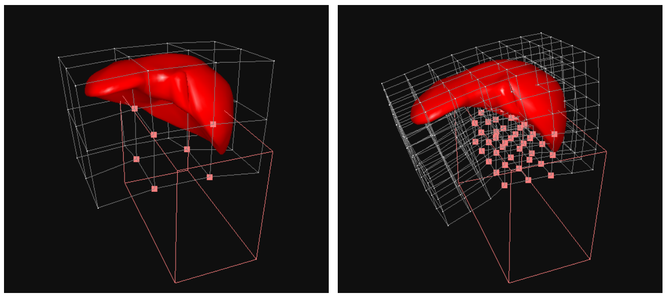


Figure 4: The liver with a grid-based mechanical model at different resolutions. The pink points are fixed.

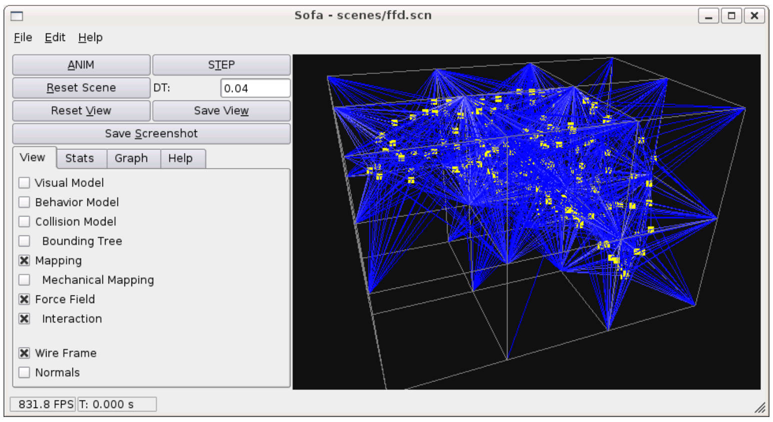
embedded in a deformable grid, whose corners are the independent degrees of freedom (DOF) of the system. The pink box defines a volume where the DOF are constrained to remain fixed. The DOF have associated mass and elastic forces.

The visual model is a geometry attached to the physical model. It is thus placed in a child node and attached using a *BarycentricMapping*. The topology and the initial vertex locations are first loaded form the given file. Then the barycentric coordinates of the vertices in the grid are computed and stored in the mapping. During the simulation, the mapping updates the visual vertex locations based on the grid corners, acting as a Free Form Deformation (FFD) operator.

This scene is written in file `liverFFD.scn` as follows:

```
<Node name="root"  dt="0.04"  showBehaviorModels="1"  showMappings="0"
      showForceFields="1">
    <Node name="LiverFFD">
        <Object type="CGImplicit"  symplectic="1"/>
        <Object type="MechanicalObject" />
        <Object type="UniformMass"  mass="1.0"  />
        <Object type="RegularGrid"
                nx="4"  ny="3"  nz="3"
                xmin="-5.25"  xmax="2.25"
                ymin="0.25"  ymax="5.25"
                zmin="-2"  zmax="3"
            />
        <Object name="GConstraint"  type="BoxConstraint"
                        box="-3.5  -2  -5  0  3  2"
                        />
        <Object type="RegularGridSpringForceField" name="Springs"
            stiffness="160" damping="4" />
        <Node name="Visu">
            <Object type="OglModel" name="Visual"
                filename="VisualModels/liver-smooth.obj" color="red" />
            <Object type="BarycentricMapping" object1="../.." object2="Visual" />
        </Node>
    </Node>
</Node>
```

The mapping from the grid corners to the vertices of the visual model is illustrated in figure 5.

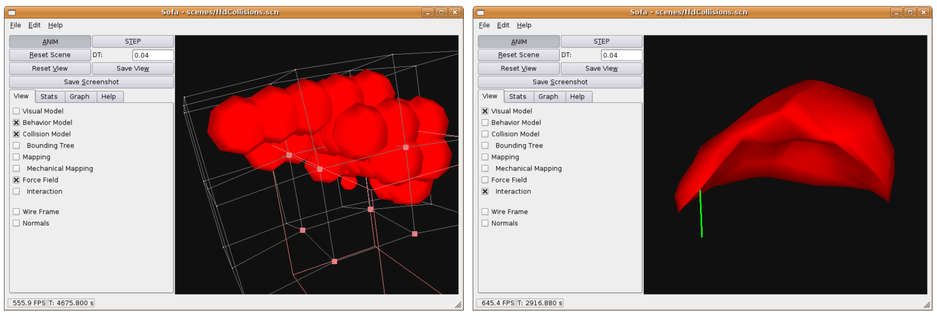


- [ ] Visual Model
- [ ] Behavior Model
- [ ] Collision Model
- [ ] Bounding Tree
- [x] Mapping
- [ ] Mechanical Mapping
- [x] Force Field
- [x] Interaction
- [x] Wire Frame
- [ ] Normals

Figure 5: The mapping (blue lines) form the corners of the deformable grid to the vertices of the visual model (yellow), visualized using the appropriate view options.

4 Interactive manipulation

We want to interactively pick and pull points of the liver. To do this, we have to attach a *CollisionModel* to the liver, as illustrated in figure 6. When



Left:
- [ ] Visual Model
- [x] Behavior Model
- [x] Collision Model
- [ ] Bounding Tree
- [ ] Mapping
- [ ] Mechanical Mapping
- [x] Force Field
- [ ] Interaction
- [ ] Wire Frame
- [ ] Normals

Right:
- [x] Visual Model
- [ ] Behavior Model
- [ ] Collision Model
- [ ] Bounding Tree
- [ ] Mapping
- [ ] Mechanical Mapping
- [ ] Force Field
- [x] Interaction
- [ ] Wire Frame
- [ ] Normals

Figure 6: Left: a *CollisionModel* made of spheres attached to the liver. Right: user interaction.

shift-clicking in the graphics window, a ray is cast from the mouse pointer through the scene, and if it intersects with a collision surface, a spring between is created. This allows to interactively manipulate the model.

The scene is modeled in file `liverFFDCollisions.scn`:

```
<Node name="root"  dt="0.04"  showBehaviorModels="1"  showMappings="0"
    showForceFields="1">
  <Object type="CollisionPipeline"  verbose="0"  />
  <Object type="BruteForceDetection"  name="N2"  />
  <Object type="VoxelGridDetection"  name="Voxel"  min="-40"  max="40"  n="8"
      draw="0"  />
  <Object type="CollisionResponse"  contact="default"  />
    <Node name="LiverFFD">
        <Object type="CGImplicit"  symplectic="1"/>
        <Object type="MechanicalObject"  />
        <Object type="UniformMass"  mass="1.0"  />
                <Object type="RegularGrid"
                        nx="8"  ny="6"  nz="6"
                        xmin="-5.25"  xmax="2.25"
                        ymin="0.25"  ymax="5.25"
                        zmin="-2"  zmax="3"
                        />
        <Object name="GConstraint"  type="BoxConstraint"
                        box="-3.5  -2  -5  0  3  2"
                        />
        <Object type="RegularGridSpringForceField"  name="Springs"
             stiffness="160"  damping="4"  />
        <Node name="Visu">
            <Object type="OglModel"  name="Visual"
                 filename="VisualModels/liver-smooth.obj"  color="red"  />
            <Object type="BarycentricMapping"  object1="../.."  object2="Visual"  />
        </Node>
                <Node name="Collision  Surface">
                  <Object type="Sphere"  name="Surf"
                       filename="CollisionModels/liver.sph"  />
                  <Object type="BarycentricMapping"  />
                </Node>
    </Node>
</Node>
```

There are two differences between this code and the version in section 3: a child node named `Collision SUrface` has been added to node `liverFFD`, and four objects related to collision detection and response have been added to node `root`.

The collision model is defined in node `Collision Surface` using two objects. Object `Surf` describes the positions, velocities and radii of the collision spheres. Their values are read from file `liver.sph` at initialization time, and then attached to the independent DOF using a *BarycentricMapping*. This mapping not only propagates positions and velocities bottom-up but also propagates forces top-down. This allows forces applied to the spheres to act on the independent DOF.

Direct interaction is set up by detecting the intersection of a pointer-based ray and the collision model. A spring between the ray is then created, and exists until the user releases the mouse button. This is achieved by the four objects added to the root node.