Solving an elasto-plastic model using DOLFIN

TTI 2005

Johan Jansson

johanjan@math.chalmers.se

Chalmers University of Technology

Overview

- Motivation
- Previous work
- Simple derivation of model
- Plasticity
- Discretization
- **FEniCS**
- Implementation using FFC/DOLFIN
- Performance results
- Future work

Motivation

Computational elasticity useful in many fields:

- Computer animation
- Computer games
- CAD
- ...

BUT, established models are crude, ad hoc (outside of CAD).

Motivation - examples

State of the art computer games use rigid body motion with joints (Half Life 2).

Motion pictures primarily use animation by hand, some cases of mass-spring simulation (hair).

Why don’t these applications use more advanced/general models?

Traditional elasticity models are difficult to understand $\Rightarrow$ difficult to apply, use effectively.

Attempt to find a simple model, attempt to automatize discretization of model.

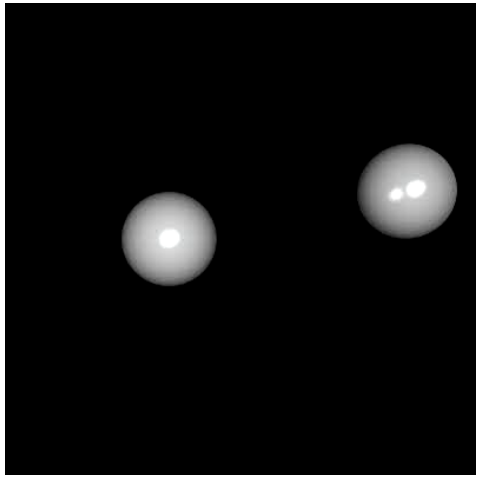
Previous work - mass-spring model

Point-masses connected by “springs”.

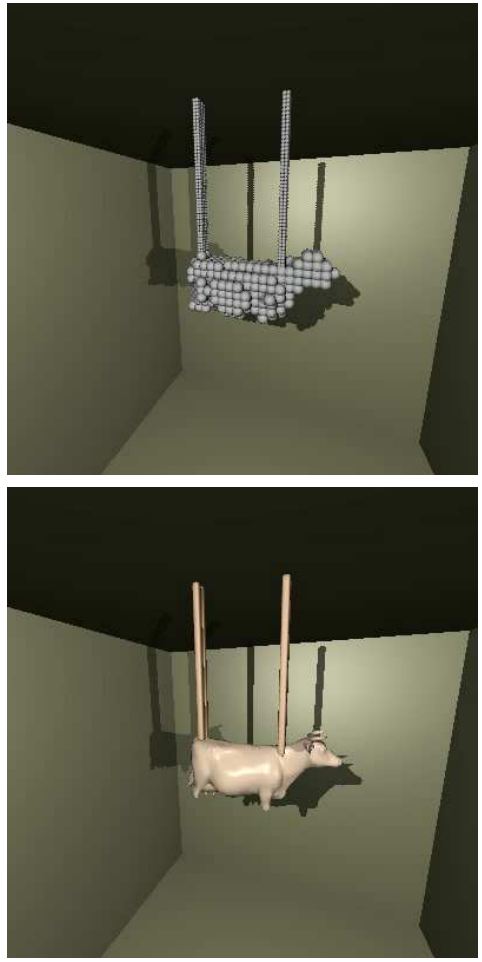
The springs represent forces between the masses - Hooke elasticity ($F = -k(e(x))$), damping ($F = -b\dot{e}(x)$) (e - elongation of spring).

Newton’s second law: ($F = m\ddot{x}$).

Collision/contact handled by each mass having a radius - add a spring when two masses intersect.



Previous work - simulations



Can we find an analogous PDE-model?

Simple derivation of model

Classical linear elasticity:

$$
\begin{aligned}
u &= x - X, \\
\dot{u} - v &= 0 \quad \text{in } \Omega^0, \\
\dot{v} - \nabla \cdot \sigma &= f \quad \text{in } \Omega^0, \\
\sigma &= E\epsilon(u) = E(\nabla u^\top + \nabla u) \\
E\epsilon &= \lambda \sum_k \epsilon_{kk} \delta_{ij} + 2\mu\epsilon, \\
v(0,\cdot) &= v^0, \quad u(0,\cdot) = u^0 \quad \text{in } \Omega^0.
\end{aligned}
$$

Only works for small displacements. Computations carried out on fixed geometry Ω^0. Why not use the deformed geometry $\Omega(t)$?

Simple derivation of model

u depends on Ω^0 (X is defined on Ω^0). Let's compute in v instead.

Differentiate σ wrt. t:

$$\sigma = E\epsilon(u) = E(\nabla u^\top + \nabla u) \Rightarrow$$
$$\dot{\sigma} = E\epsilon(v) = E(\nabla v^\top + \nabla v)$$

Rest of the model remains the same.

The elastic model

Now formulate the model in the deformed geometry $\Omega(t)$:

$$\begin{aligned} \dot{u} - v &= 0 \quad \text{in } \Omega(t), \\ \dot{v} - \nabla \cdot \sigma &= f \quad \text{in } \Omega(t), \\ \dot{\sigma} &= E\epsilon(v) = E(\nabla v^\top + \nabla v) \\ v(0,\cdot) = v^0, &\quad u(0,\cdot) = u^0 \quad \text{in } \Omega^0. \end{aligned}$$

The model is a piecewise linear elastic model. Given some geometry Ω_i we compute using the linear model (small displacements) for a small time step and produce the geometry Ω_{i+1}. The process is then repeated.

Examples

Elastic bar (Updated Lagrange)
Elastic bar (Mass-spring)
Elastic cube (Classical elasticity)

Viscosity

$$\dot{v} - \nabla \cdot \sigma - \nu\epsilon(v) = f \quad \text{in } \Omega(t)$$

We add a simple viscous term to model viscosity in materials.

Plasticity

$$\dot{\sigma} = E(\epsilon(v) - \frac{1}{\nu_p}(\sigma - \pi\sigma)) \quad \text{in } \Omega(t),$$

$$\pi\sigma = \frac{\sigma}{\|\sigma\|}, \|\sigma\| > Y_s$$

$$\pi\sigma = \sigma, \|\sigma\| \leq Y_s$$

Visco-plastic model. $\pi\sigma$ is the projection on to the set of admissible stresses. Y_s is the yield stress of the material.

Examples (Plasticity)

Elastic diamond
Plastic diamond
Plastic bar (no gravity)

Discretization of model

Finite element discretization (elastic model):

$$\dot{v} - \nabla \cdot \sigma = f \Rightarrow$$

$$\int_\Omega \dot{v} q \, dx - \int_\Omega \nabla \cdot \sigma q \, dx = \int_\Omega f q \, dx$$

Ansatz:

$$v_h = \sum_j \xi_j^v \phi_j$$

$$\sigma_h = \sum_j \xi_j^\sigma \psi_j$$

ϕ piecewise linear. ψ piecewise constant.

Discretization of model

Insert ansatz and choose test space as piecewise linear:

$$\int_\Omega \sum_j \dot{\xi}_j^v \phi_j \phi_i \, dx - \int_\Omega \sum_j \xi_j^\sigma \psi_j \nabla \phi_i \, dx = \int_\Omega f \phi_i \, dx, \forall i \Rightarrow$$

$$M\xi^v - A\xi^\sigma = b$$

Discretization of model

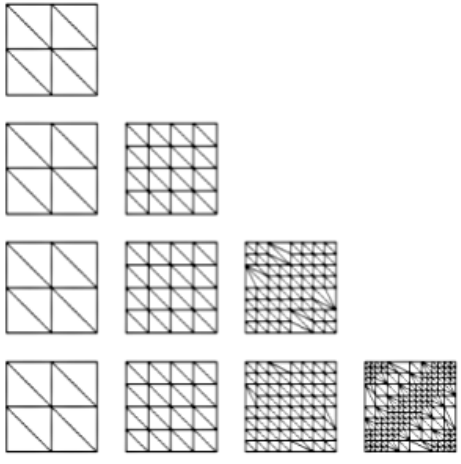
$$\dot{\sigma}_h^{K_i} = E(\nabla v^\top + \nabla v)^{K_i}, \forall K_i \in T_\Omega(t)$$

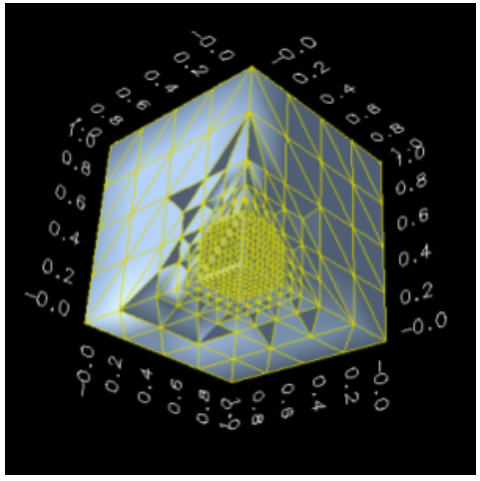
The FEniCS project

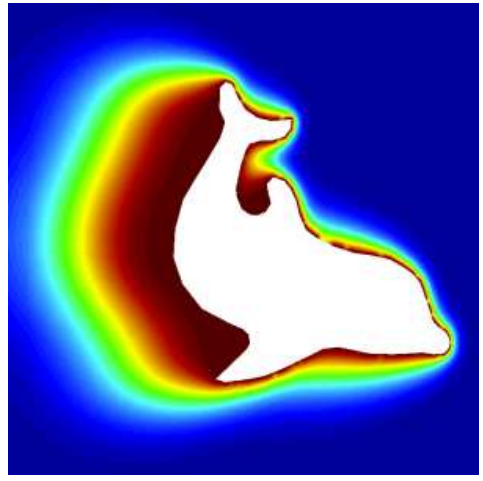
A free software project for the Automation of Computational Mathematical Modeling (ACMM), developed at

- The Toyota Technological Institute at Chicago
- The University of Chicago
- Chalmers University of Technology (Göteborg, Sweden)
- Argonne National Laboratory (Chicago)

More information at `http://www.fenics.org/`







Projects

- **DOLFIN**, the C++ interface of FEniCS
  - Hoffman, Jansson, Logg, et al.
- **FErari**, optimized form evaluation
  - Kirby, Knepley, Scott
- **FFC**, the FEniCS Form Compiler
  - Logg
- **FIAT**, automatic generation of finite elements
  - Kirby, Knepley
- **Ko**, simulation of mechanical systems
  - Jansson
- **Puffin**, light-weight version for Octave/MATLAB
  - Hoffman, Logg

Solving a PDE with DOLFIN

Implementing a solver

```
void PoissonSolver::solve()
{
  Galerkin     fem;
  Matrix       A;
  Vector       x, b;
  Function     u(mesh, x);
  Function     f(mesh, "source");
  Poisson      poisson(f);
  KrylovSolver solver;
  File         file("poisson.m");

  fem.assemble(poisson, mesh, A, b);
  solver.solve(A, x, b);

  u.rename("u", "temperature");
  file << u;
}
```

Automatic assembling

```
class Poisson : public PDE {
  ...
  real lhs(const ShapeFunction& u, const ShapeFunction& v)
  {
    return (grad(u),grad(v)) * dK;
  }

  real rhs(const ShapeFunction& v)
  {
    return f*v * dK;
  }
  ...
};
```

Automatic assembling

```
class ConvDiff : public PDE {
  ...
  real lhs(const ShapeFunction& u, const ShapeFunction& v)
  {
    return (u*v + k*((b,grad(u))*v + a*(grad(u),grad(v))))*dK;
  }

  real rhs(const ShapeFunction& v)
  {
    return (up*v + k*f*v) * dK;
  }
  ...
};
```

Abstract assembly algorithm

$$
\begin{aligned}
(A_h)_{ij} &= a(\varphi_j, \hat{\varphi}_i) = \int_\Omega A(\varphi_j)\hat{\varphi}_i \, dx \\
&= \sum_{K \in \mathcal{T}} \int_K A(\varphi_j)\hat{\varphi}_i \, dx = \sum_{K \in \mathcal{T}} a(\varphi_j, \hat{\varphi}_i)_K .
\end{aligned}
$$

Iterate over all elements K and for each element K compute the contributions to all $(A_h)_{ij}$, for which φ_j and $\hat{\varphi}_i$ are supported within K.

Assemble on reference element and map values to real element.

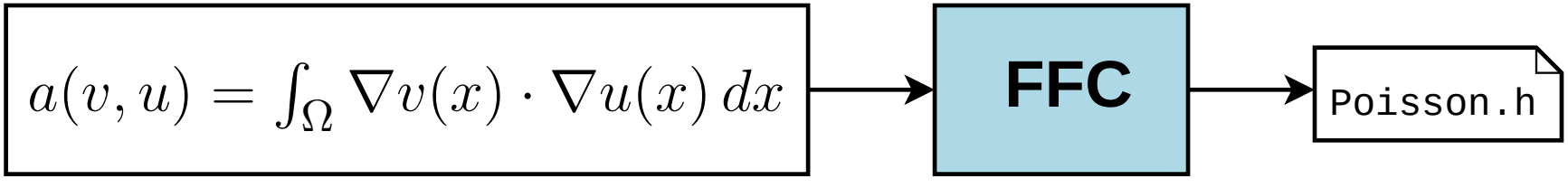
The elasto-plastic model in DOLFIN

```
class ElasticityUpdated : public PDE {

  ...

  real rhs(ShapeFunction::Vector& v)
  {
    ... Compute sigma ...

     return (k * (f(0) * v(0) + f(1) * v(1) + f(2) * v(2)) -
               k * (sigma(0, 0) * v(0).ddx() +
                    sigma(0, 1) * v(0).ddy() +
                    sigma(0, 2) * v(0).ddz() +
                    sigma(1, 0) * v(1).ddx() +
                    sigma(1, 1) * v(1).ddy() +
                    sigma(1, 2) * v(1).ddz() +
                    sigma(2, 0) * v(2).ddx() +
                    sigma(2, 1) * v(2).ddy() +
                    sigma(2, 2) * v(2).ddz())) * dx;
  }
};
```

Solving a PDE with FFC/DOLFIN

FFC: the FEniCS Form Compiler

- Automates a key step in the implementation of finite element methods for partial differential equations
- Input: a variational form and a finite element
- Output: optimal C/C++ code




```
>> ffc [-l language] poisson.form
```

Example: Classical Elasticity

The form:

$$a(u, v) = \int_\Omega \sigma(u)\epsilon(v)$$
$$l(v) = \int_\Omega fv$$

where (as before):

$$\epsilon(u) = \nabla u^\top + \nabla u$$
$$\sigma(u) = E\epsilon(u)$$
$$E\epsilon = \lambda \sum_k \epsilon_{kk}\delta_{ij} + 2\mu\epsilon,$$

Example: Classical Elasticity

FFC representation (`Elasticity.form`):

```
# The bilinear form for classical linear elasticity
# Compile this form with FFC: ffc Elasticity.form.

name = "Elasticity"
element = FiniteElement("Lagrange", "tetrahedron", 1, 3)

c1 = Constant() # Lame coefficient
c2 = Constant() # Lame coefficient
f = Function(element) # Source

v = BasisFunction(element)
u = BasisFunction(element)

a = (2.0 * c1 * u[i].dx(i) * v[j].dx(j) +
c2 * (u[i].dx(j) + u[j].dx(i)) * (v[i].dx(j) + v[j].dx(i))) * dx
L = f[i] * v[i] * dx
```

Example: Classical Elasticity

FFC output (`Elasticity.h`):

```
BilinearForm(const real& c0, const real& c1) : ...

bool interior(real* block) const
{
  // Compute geometry tensors
  real G0_0_0_0_0 = det*c0*g00*g00;
  real G0_0_0_0_1 = det*c0*g00*g10;
  ...

  // Compute element tensor
  block[0] =
  3.333333333333329e-01*G0_0_0_0_0 + 3.333333333333329e-01*G0_0_0_0_1 +
  3.333333333333329e-01*G0_0_0_0_2 + 3.333333333333329e-01*G0_0_0_1_0 +
  3.333333333333329e-01*G0_0_0_1_1 + 3.333333333333329e-01*G0_0_0_1_2 +
  3.333333333333329e-01*G0_0_0_2_0 + 3.333333333333329e-01*G0_0_0_2_1 +
  3.333333333333329e-01*G0_0_0_2_2 + 1.666666666666664e-01*G1_0_0 +
  1.666666666666664e-01*G1_0_1 + 1.666666666666664e-01*G1_0_2 +
  1.666666666666664e-01*G1_1_0 + 1.666666666666664e-01*G1_1_1 +
  ...
```

Benchmark

Beam example

Task: assemble global matrix and load vector

Benchmark results

- Mesh 1
  24576 (25k) cells

  old:
  matrix - 19.2s
  vector - 5.21s

  ffc:
  matrix - 0.770s
  vector - 0.0900s

  speedup:
  matrix - 25
  vector - 58

Benchmark results

- Mesh 2
  162000 (162k) cells

  old:
  matrix - 119s
  vector - 33.2s

  ffc:
  matrix - 9.12s
  vector - 0.560s

  speedup:
  matrix - 13
  vector - 60

Future work

Solve elasto-plastic model using FFC

Multi-adaptive time stepping

Space adaptivity

Interface to fluid mechanics (Navier-Stokes).