ArtiSynth Tutorial

PMHA 2014

University of British Columbia,
Vancouver, Canada

Program

14:00 Welcome and introductions
14:10 ArtiSynth overview (John Lloyd)
14:30 Model building tutorial

* John Lloyd: Basic mechanical models
* Ian Stavness: Simulation control
* Antonio Sanchez: Adding FEM models

15:40 Break
16:00 Short presentations by other developers
16:20 General questions and feedback
16:50 Breakout sessions

* Help with installing and compiling
* Getting your model into ArtiSynth
* Specific project questions

ArtiSynth: A Toolkit Combining MultiBody and Finite Element Simulation

John Lloyd

University of British Columbia, Vancouver, Canada

ArtiSynth

An open source system for mechanical simulation combining multibody and FEM capabilites

Targeted at applications in biomechanics, physiology, medicine, and dentisry

Possible applications: treatment planning and prediction, ergonomic design

Motivations

- Hard to find all required features in current systems
- Ability to implement and test new techniques
- Need for interactivity

ArtiSynth

Written in Java, using JNI to OpenGL and linear solvers (Pardiso); can call from Matlab

Java API for model creation, used to generate most of the models (biomechanical engineers)

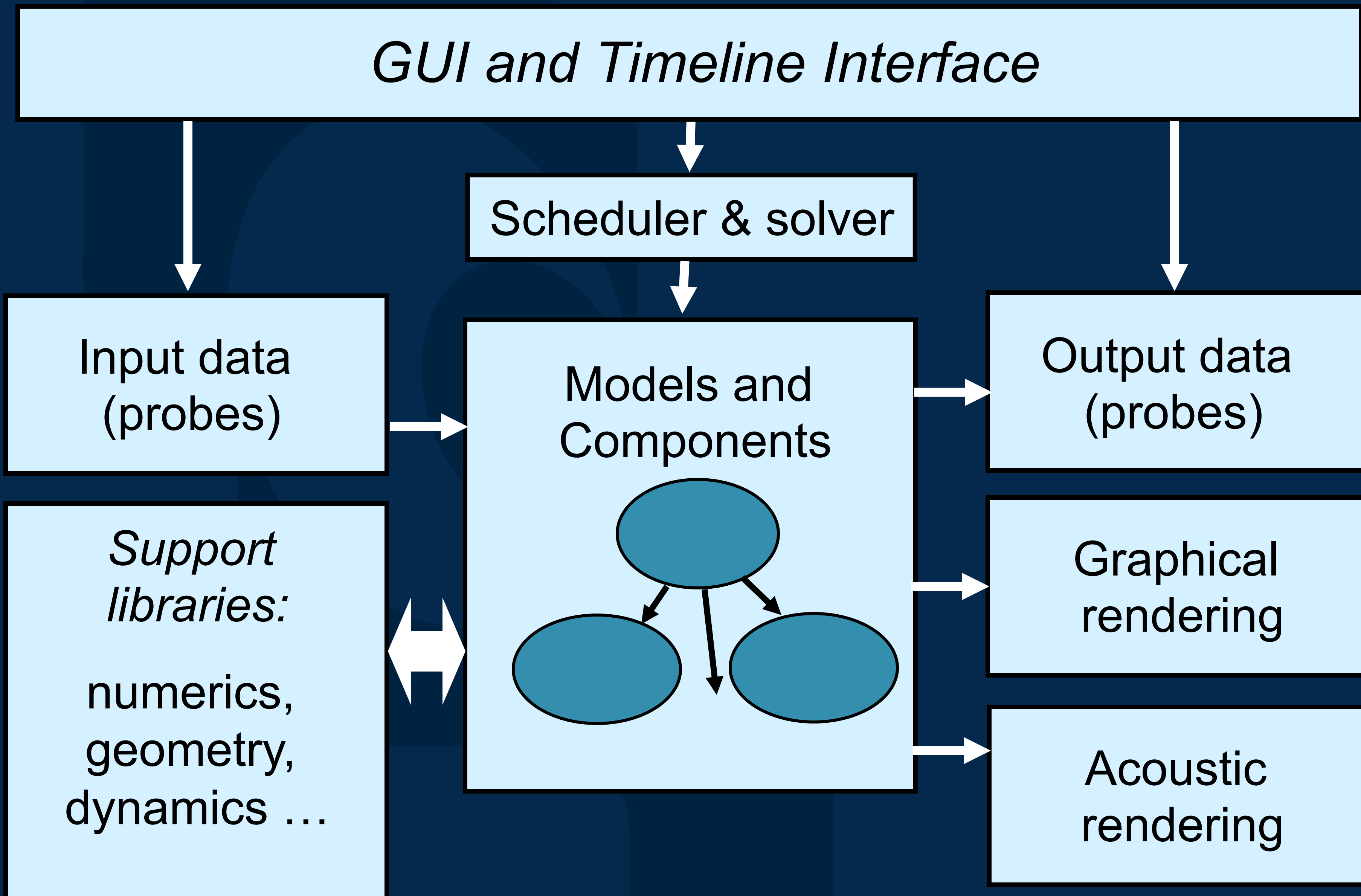
Graphical interface for interactive editing, simulation, and observation (scientists, clinicians)

ArtiSynth Demo Reel

www.artisynth.org

July 2014

System structure


GUI and Timeline Interface
Scheduler & solver
Input data (probes)
Models and Components
Output data (probes)
Graphical rendering
Acoustic rendering
Support libraries:
numerics, geometry, dynamics …

Model Component Hierarchy

- RootModel v
    - models v
        - MechModel v
            - particles >
            - springs >
            - rigidBodies >
            - rigidBodyConnectors >
            - . . .
    - inputProbes >
    - controllers >
    - monitors >
    - outputProbes >
    - controlPanels >

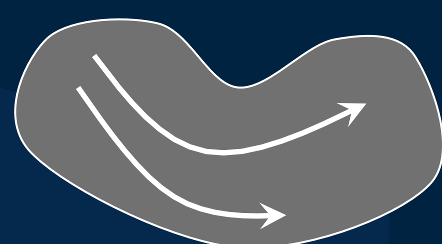
Mechanical components

Dynamic components

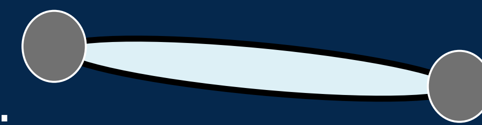
particles (3 DOF)

rigid bodies (6 DOF)

reduced coordinates (r DOF)

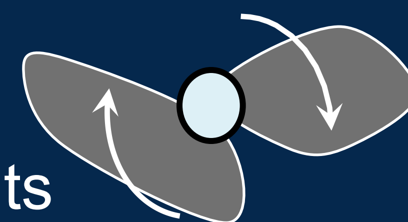


Force effectors

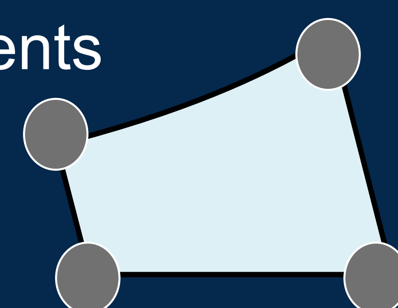


axial springs

constraints



finite elements



Per-step model advancement

```
t1 = t0 + h;
applyInputProbes (model, t1);
applyControllers (model, t0, t1);
model.advance (t0, t1);
applyMonitors (model, t0, t1);
applyOutputProbes (model, t1);
t0 = t1;
```

Properties

Components can have class-specific *properties:*

Particle

mass
position
velocity

AxialSpring

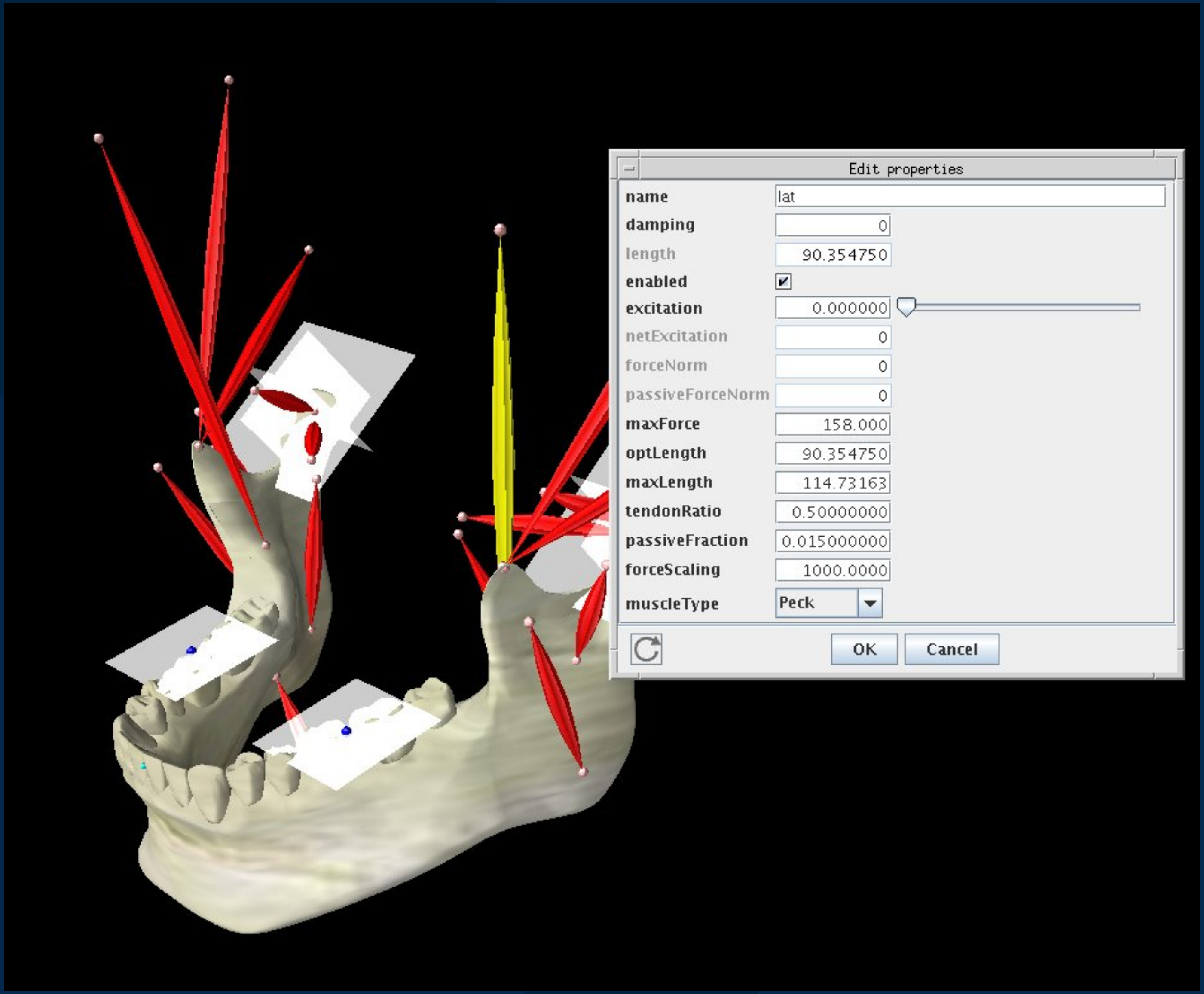
stiffness
damping
restLength

Properties exported in class definition

```
static {
   myProps.add ("mass", "particle mass", 1);
}

public double getMass() {
   return myMass;
}

public void setMass(double m) {
   myMass = m;
}
```

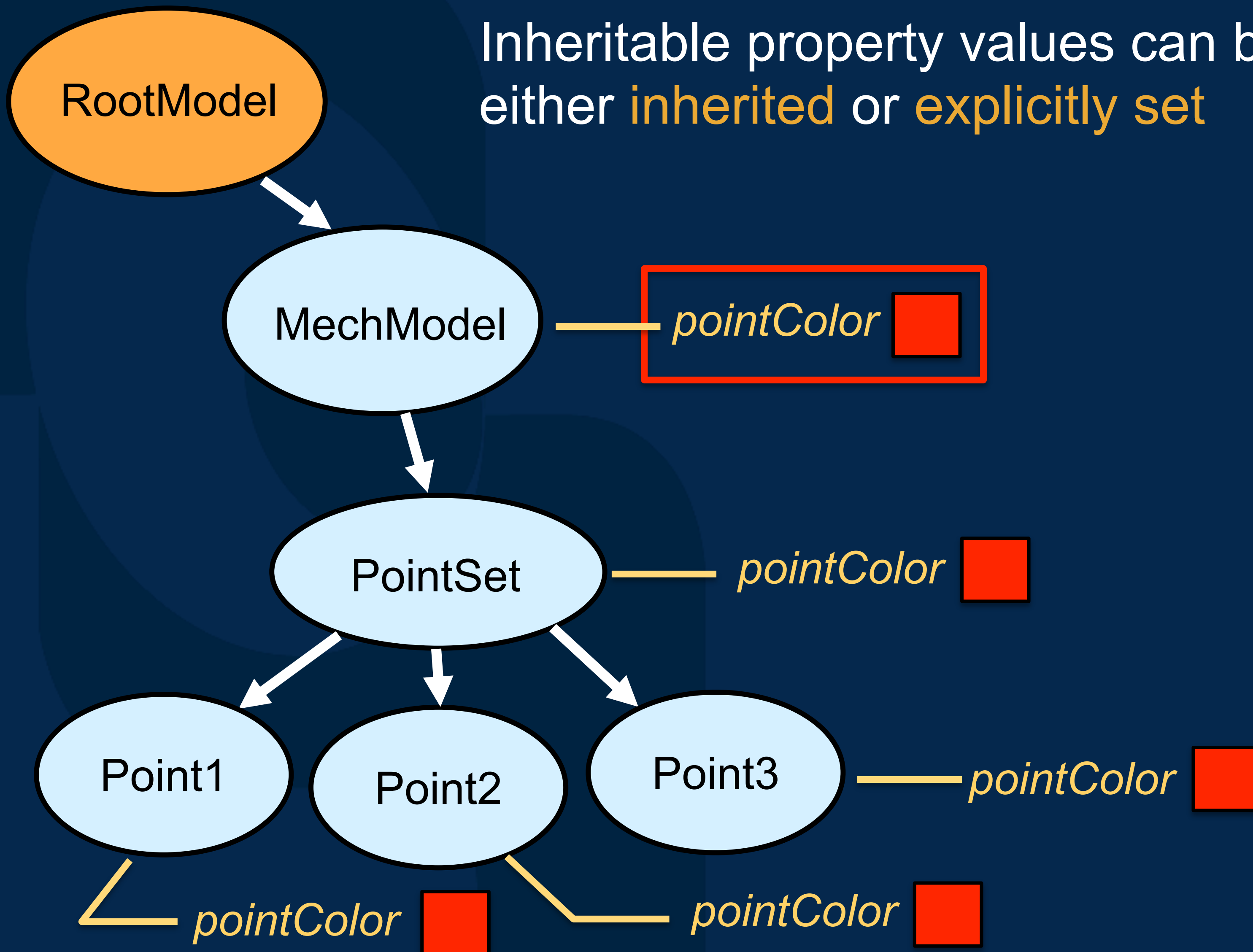
- Class reflection used to determine attributes
- Properties then used to automatically create control panels and probes, read/write files, etc.

Interactive property editing



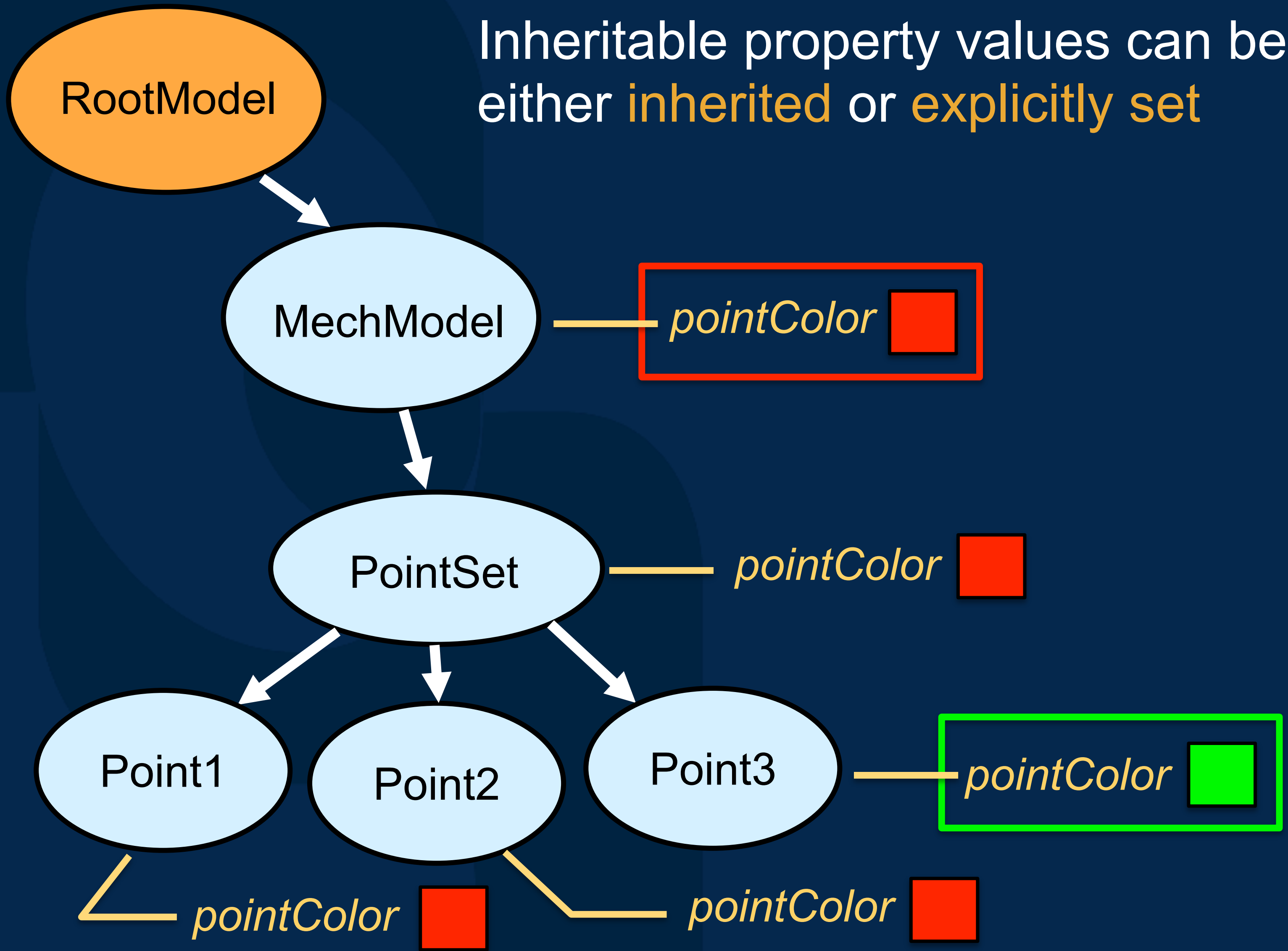
Properties can be made *inheritable*

Inheritable property values can be either inherited or explicitly set



Properties can be made *inheritable*

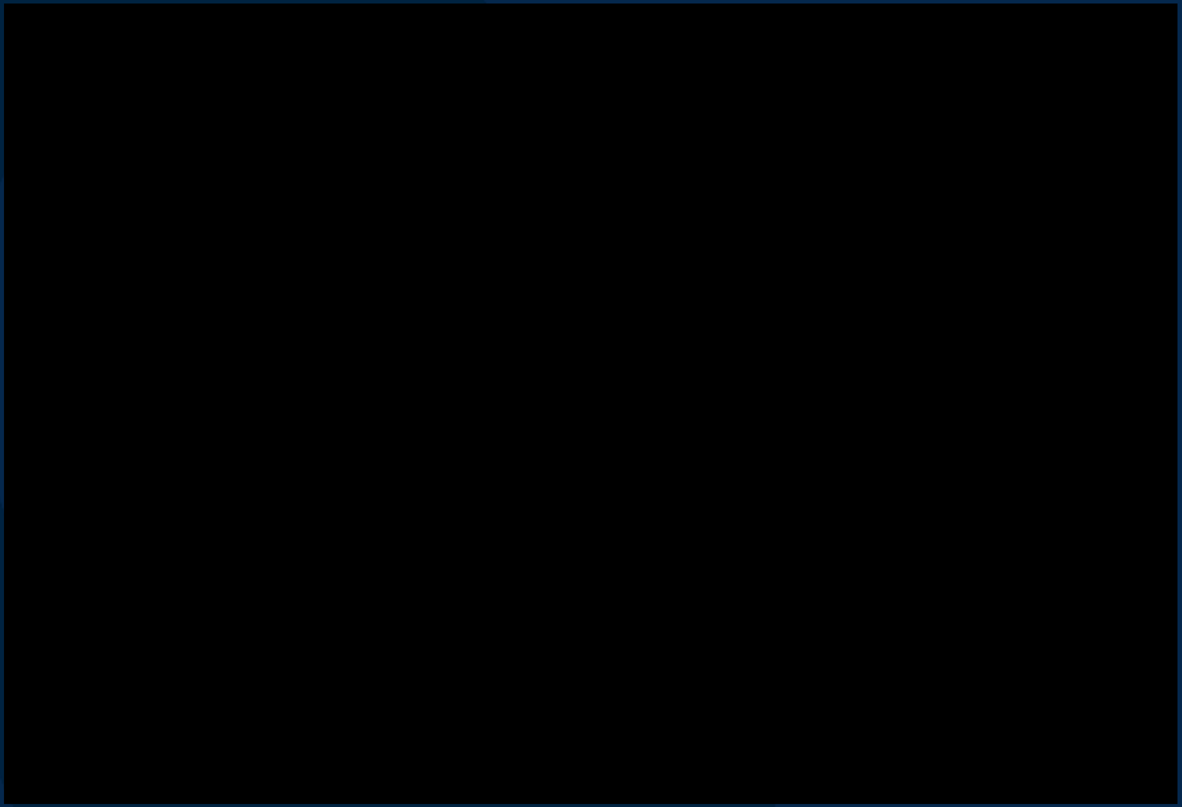
Inheritable property values can be either inherited or explicitly set



Physics simulation

ArtiSynth is essentially a multi-body simulator with support for finite elements and deformable bodies

Full coordinate formulation



Combined system state: positions q, velocities u

Combined system mass: M

Apply forces:

$$f = \frac{d(Mu)}{dt} = Ma + \dot{M}u$$

$$a = M^{-1}\bar{f}, \ \bar{f} \equiv f - \dot{M}u$$

Discrete solution:

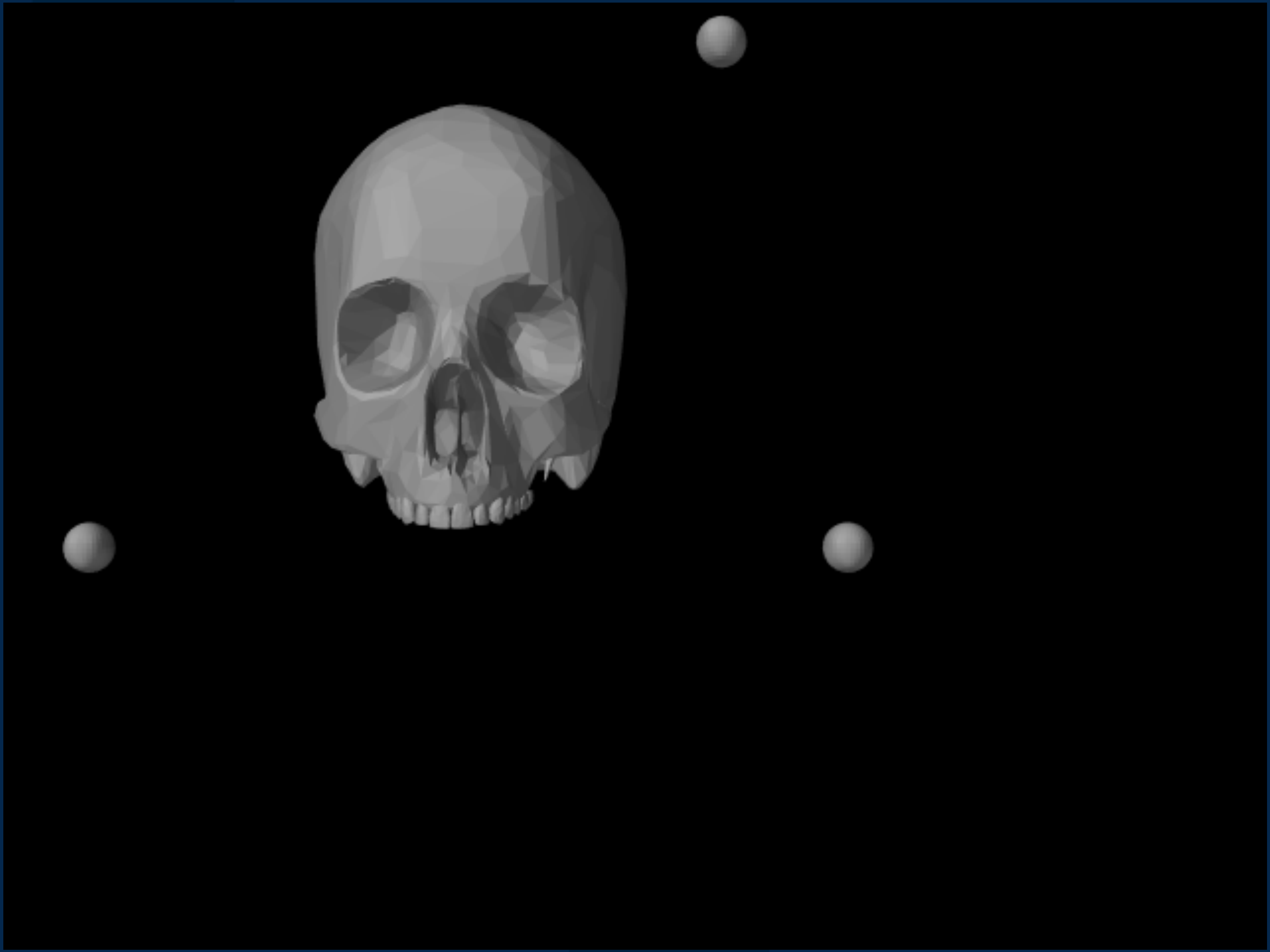
$$u^{k+1} = u^k + ha$$

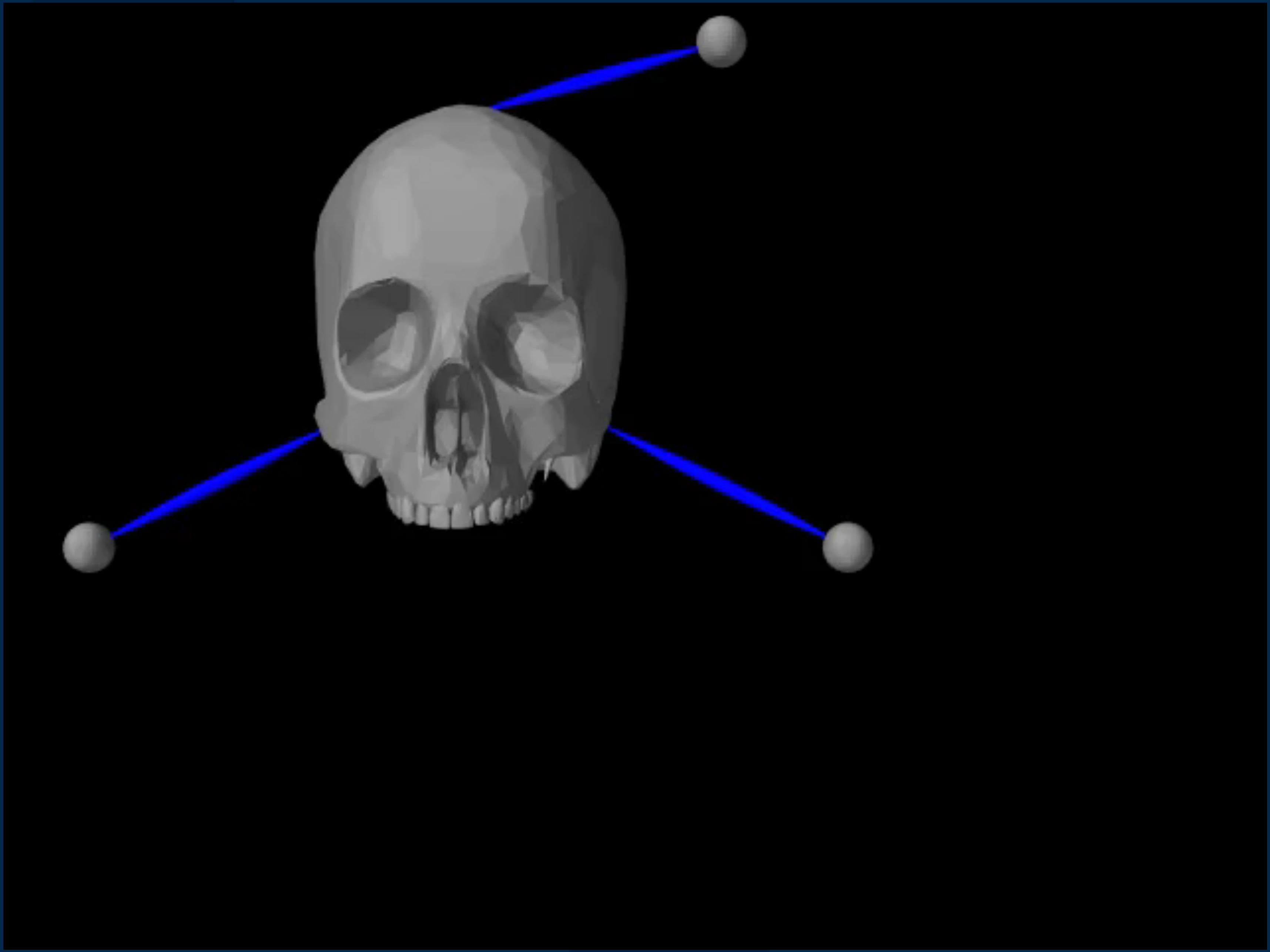
Integration

$$u^{k+1} = u^k + hM^{-1}\bar{f}$$

$$\boxed{\begin{aligned} Mu^{k+1} &= Mu^k + h\bar{f} \\ q^{k+1} &= q^k + hu^{k+1} \end{aligned}}$$

Symplectic Euler Integrator





Bilateral Constraints

Example: particle kept on a surface

$0 = g \cdot u$ enforced by: $f_c = g^T \lambda$

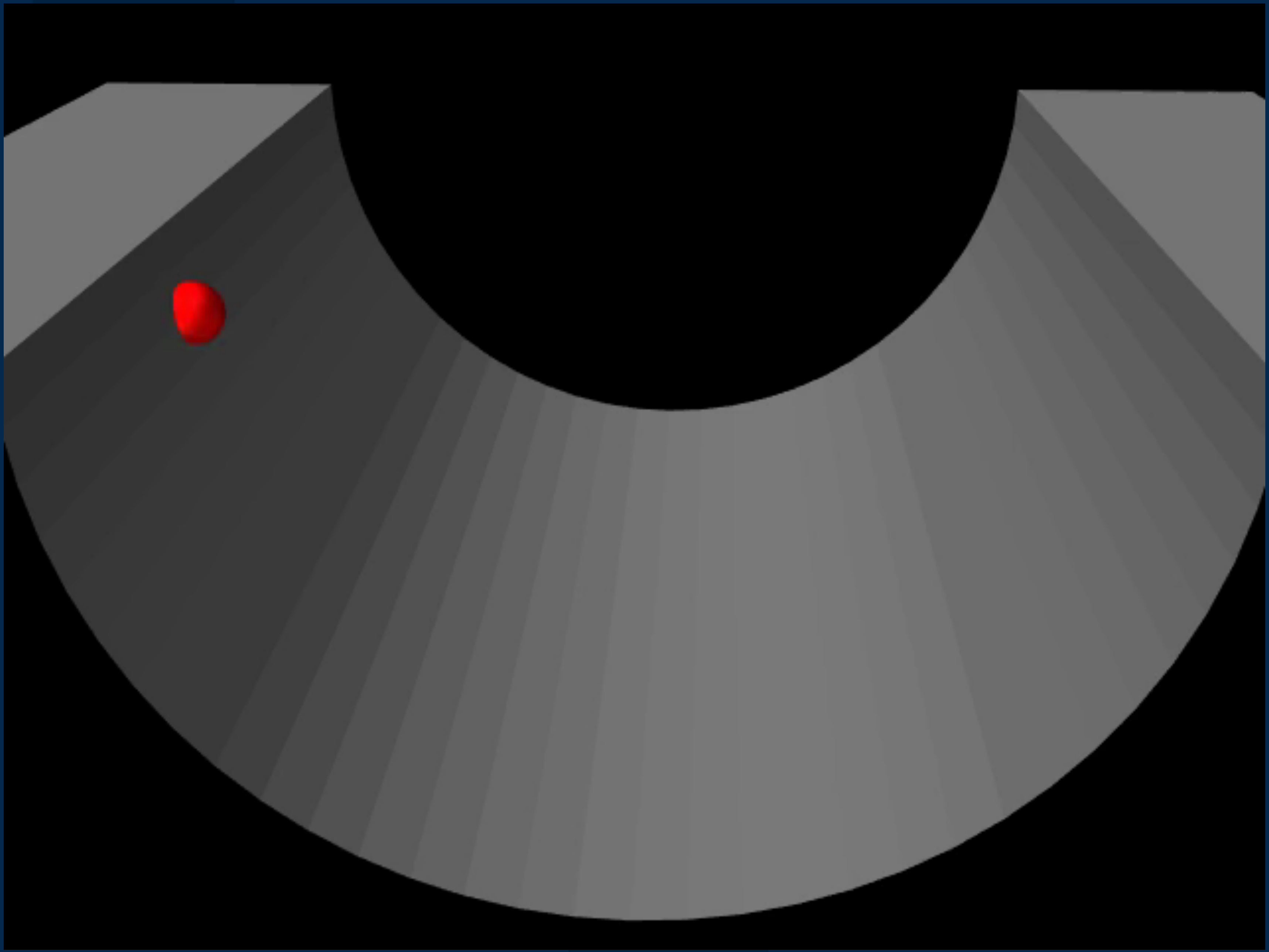
Integration augmented with constraints:

$$Mu^{k+1} = Mu^k + h\bar{f}^k + G^T\lambda$$

$$Gu^{k+1} = 0$$

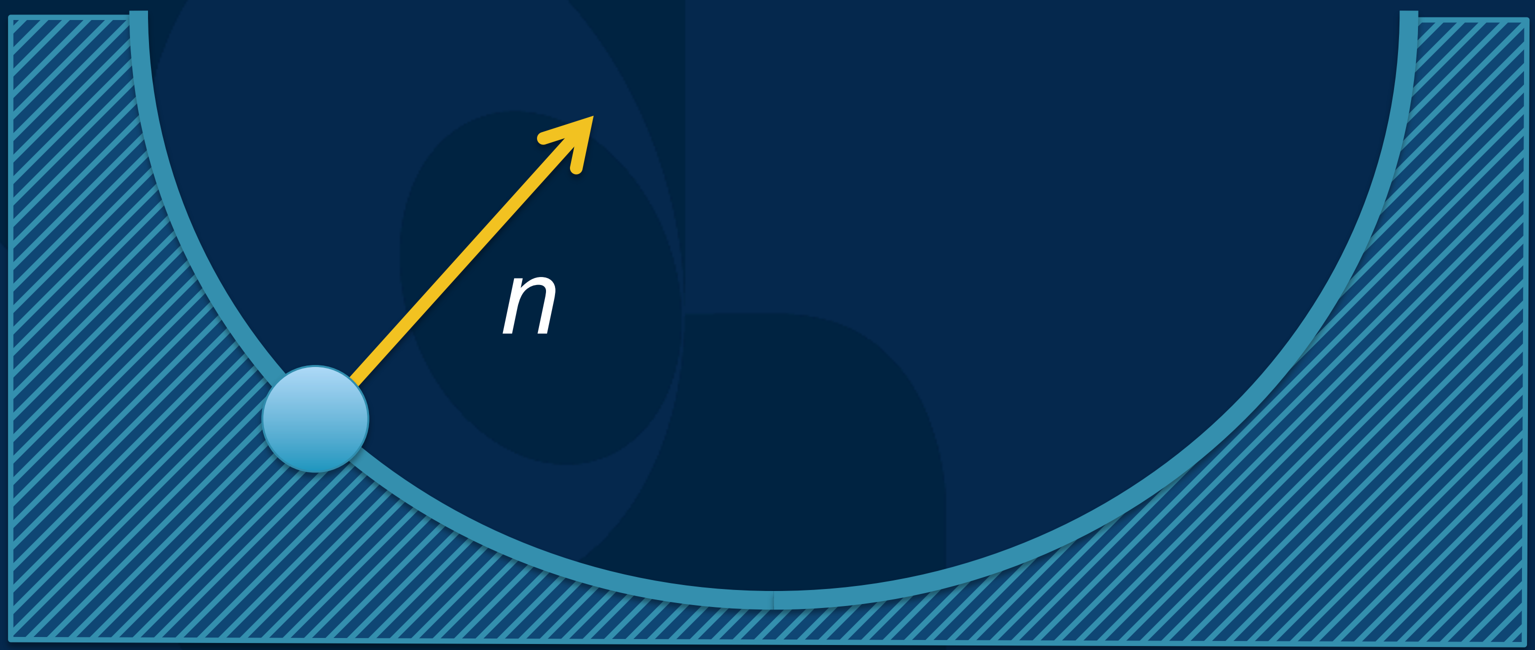
Can be arranged as a KKT system:

$$\begin{pmatrix} M & -G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} u^{k+1} \\ \lambda \end{pmatrix} = \begin{pmatrix} b_f \\ b_g \end{pmatrix}$$



Unilateral Constraints (Contact)

Example: particle contacting a surface



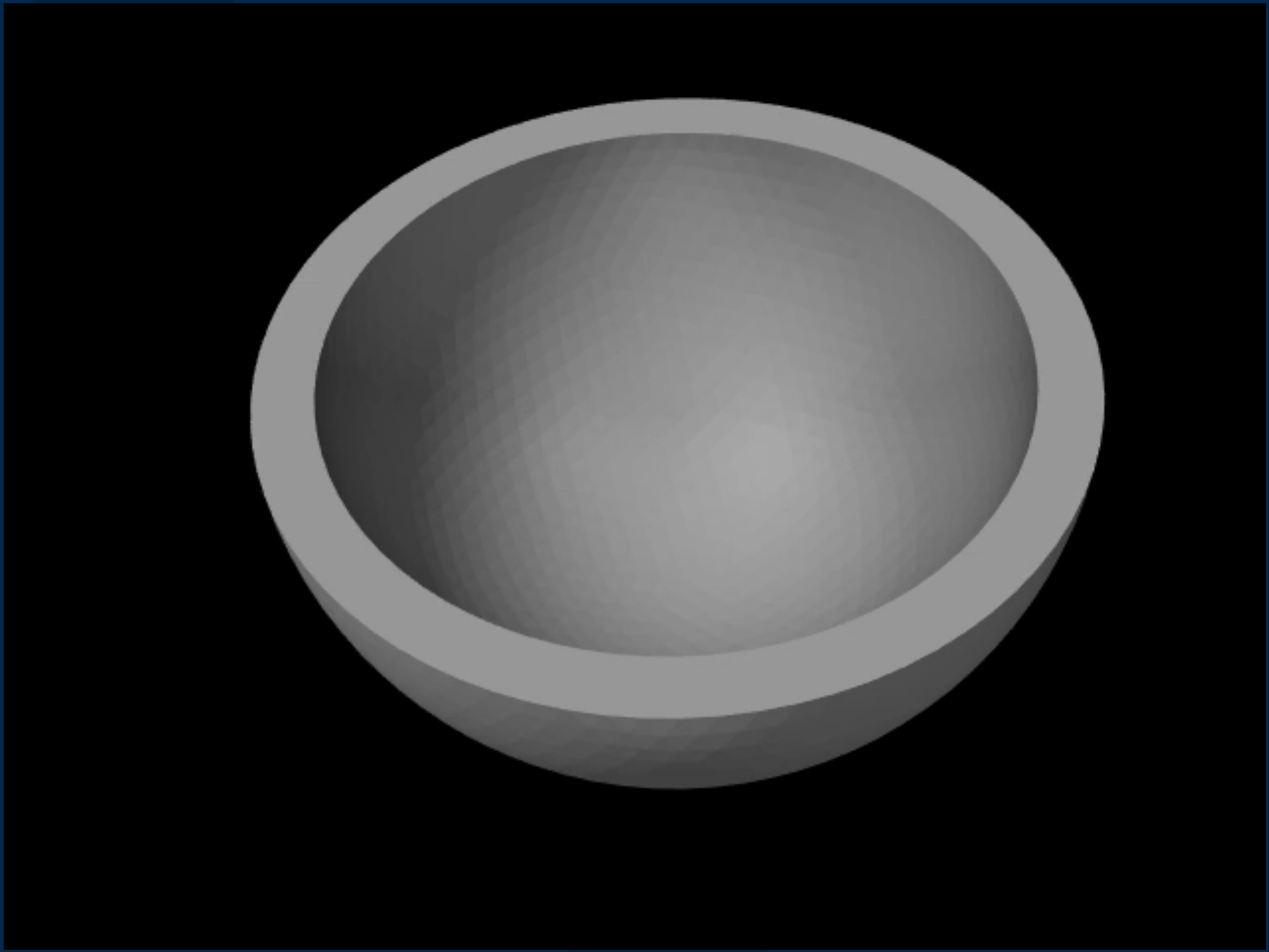
$$0 \leq n \cdot u \quad \perp \quad n^T \lambda \geq 0$$

Leads to a Mixed Linear Complementarity Problem (MLCP):

$$\begin{pmatrix} M & -G^T & -N^T \\ G & 0 & 0 \\ N & 0 & 0 \end{pmatrix} \begin{pmatrix} u^{k+1} \\ \lambda \\ z \end{pmatrix} + \begin{pmatrix} -b_f \\ -b_g \\ -b_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ w \end{pmatrix}$$

$$0 \leq z \perp w \geq 0$$

Same system arises in higher order integrators (e.g., Newmark methods)



FEM Coupling

1. *Requires implicit integration*
2. *Nodes are treated as particles*
3. *Provide attachments between components*

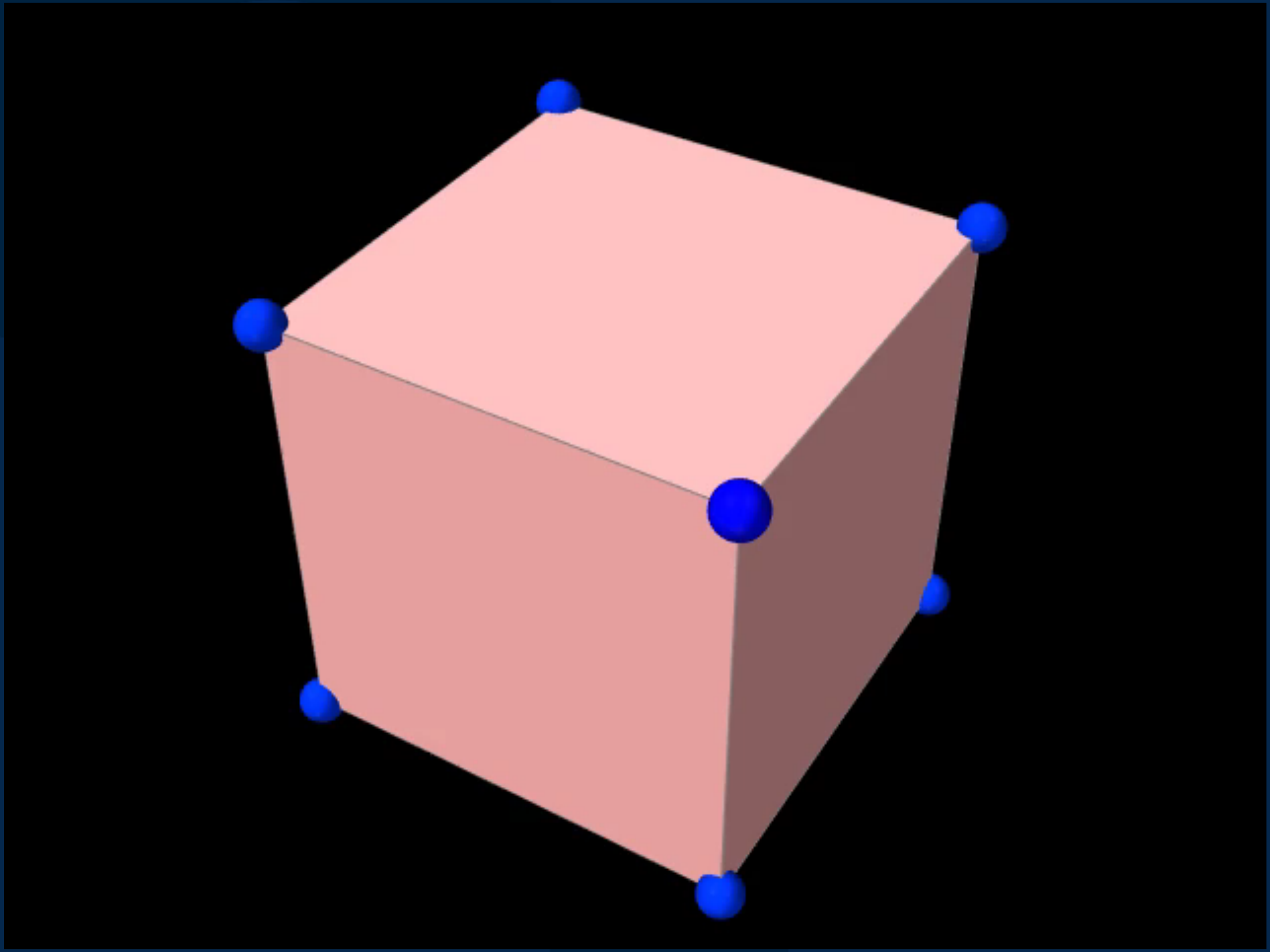
Implicit integration: modified MLCP :

$$\begin{pmatrix} \boxed{\hat{M}} & -G^T & -N^T \\ G & 0 & 0 \\ N & 0 & 0 \end{pmatrix} \begin{pmatrix} u^{k+1} \\ \lambda \\ z \end{pmatrix} + \begin{pmatrix} \boxed{-\hat{b}_f} \\ -b_g \\ -b_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ w \end{pmatrix}$$

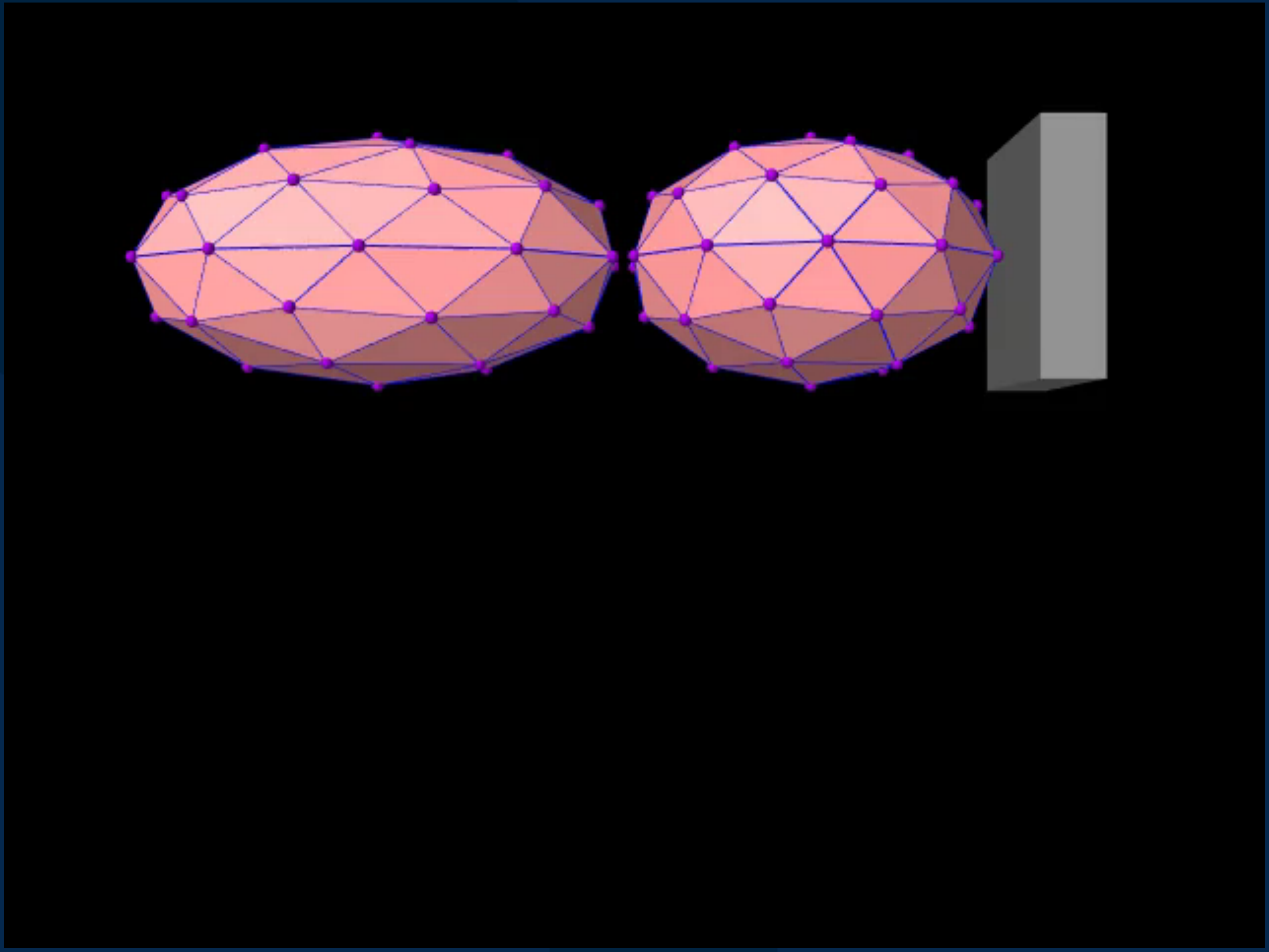
$$0 \leq z \perp w \geq 0$$

[Anitescu & Hart, 2004; Potra et. al. 2006]

Nodes as particles: elements like volumetric springs



Attachments between components:



Questions?

Modeling Building Tutorial: Basic Mechanical Models

Creating a model in Java

- Create a subclass of RootModel
- Override the build() method to create and assemble model components

```
package artisynth.models.xxx;
import artisynth.core.workspace.RootModel;
import artisynth.core.mechmodels.*;

public class MyModel extends RootModel {

   public void build(String[] args) {
        MechModel mech = new MechModel("mech");
        addModel (mech);
        ... create components for mech ...
   }
}
```

Model Component Hierarchy

- RootModel v
  - models v
    - MechModel v
      - particles >
      - springs >
      - rigidBodies >
      - rigidBodyConnectors >
      - . . .
  - inputProbes >
  - controllers >
  - monitors >
  - outputProbes >
  - controlPanels >

artisynth.demos.tutorial.ParticleSpring

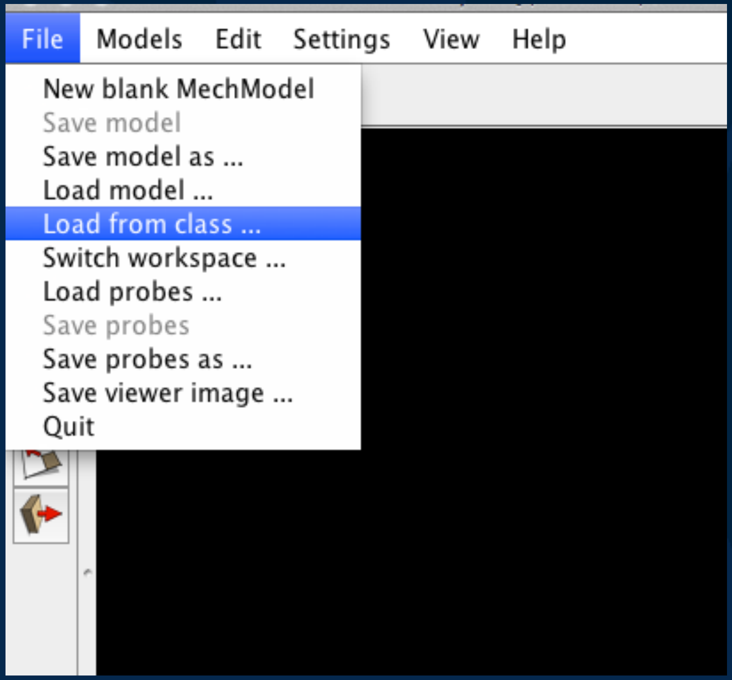
```
public class ParticleSpring extends RootModel {

   public void build (String[] args) {

      // create MechModel and add to RootModel
      MechModel mech = new MechModel ("mech");
      addModel (mech);

      // create the components
      Particle p1 = new Particle ("p1", /*mass=*/2, 0, 0, 0);
      Particle p2 = new Particle ("p2", /*mass=*/2, 1, 0, 0);
      AxialSpring spring = new AxialSpring ("spr", /*restLength=*/0);
      spring.setPoints (p1, p2);
      spring.setMaterial (
         new LinearAxialMaterial (/*stiffness=*/20, /*damping=*/10));

      // add components to the mech model
      mech.addParticle (p1);
      mech.addParticle (p2);
      mech.addAxialSpring (spring);

```

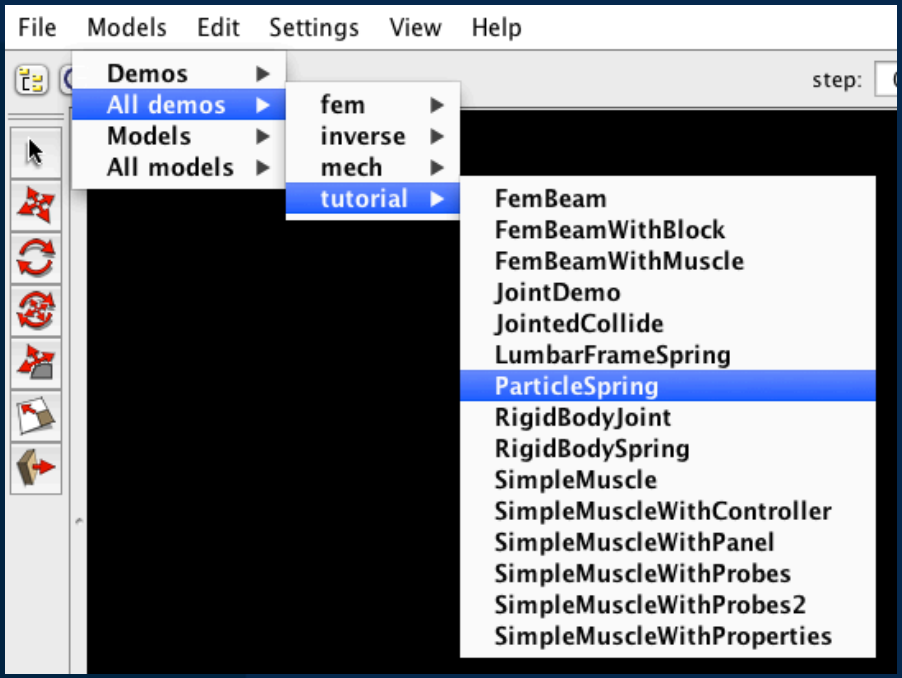
ParticleSpring (cont)

```java

      p1.setDynamic (false);                    // first particle set to be fixed
      mech.setBounds (-1, 0, -1, 1, 0, 0);  // increase viewer bounds

      // set render properties for the components
      setPointRenderProps (p1);
      setPointRenderProps (p2);
      setSpringRenderProps (spring);
   }

   protected void setPointRenderProps (Point p) {
      RenderProps.setPointColor (p, Color.RED);
      RenderProps.setPointStyle (p, RenderProps.PointStyle.SPHERE);
      RenderProps.setPointRadius (p, 0.06);
   }

   protected void setSpringRenderProps (AxialSpring s) {
      RenderProps.setLineColor (s, Color.BLUE);
      RenderProps.setLineStyle (s, RenderProps.LineStyle.CYLINDER);
      RenderProps.setLineRadius (s, 0.02);
   }
```

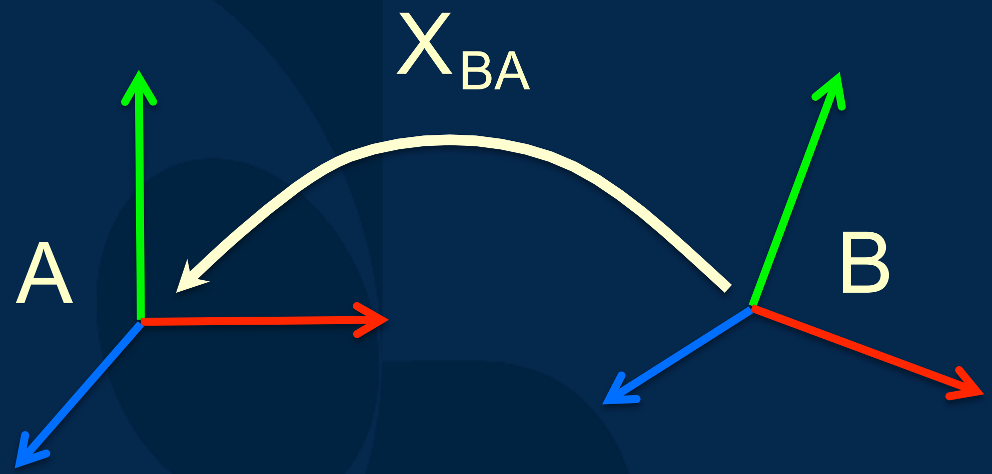
Loading the model into ArtiSynth

- Make sure the model is in ArtiSynth's classpath:
  Link model project in eclipse, or
  Specify model classpath in EXTCLASSPATH file
- Load explicitly or use the model menu:



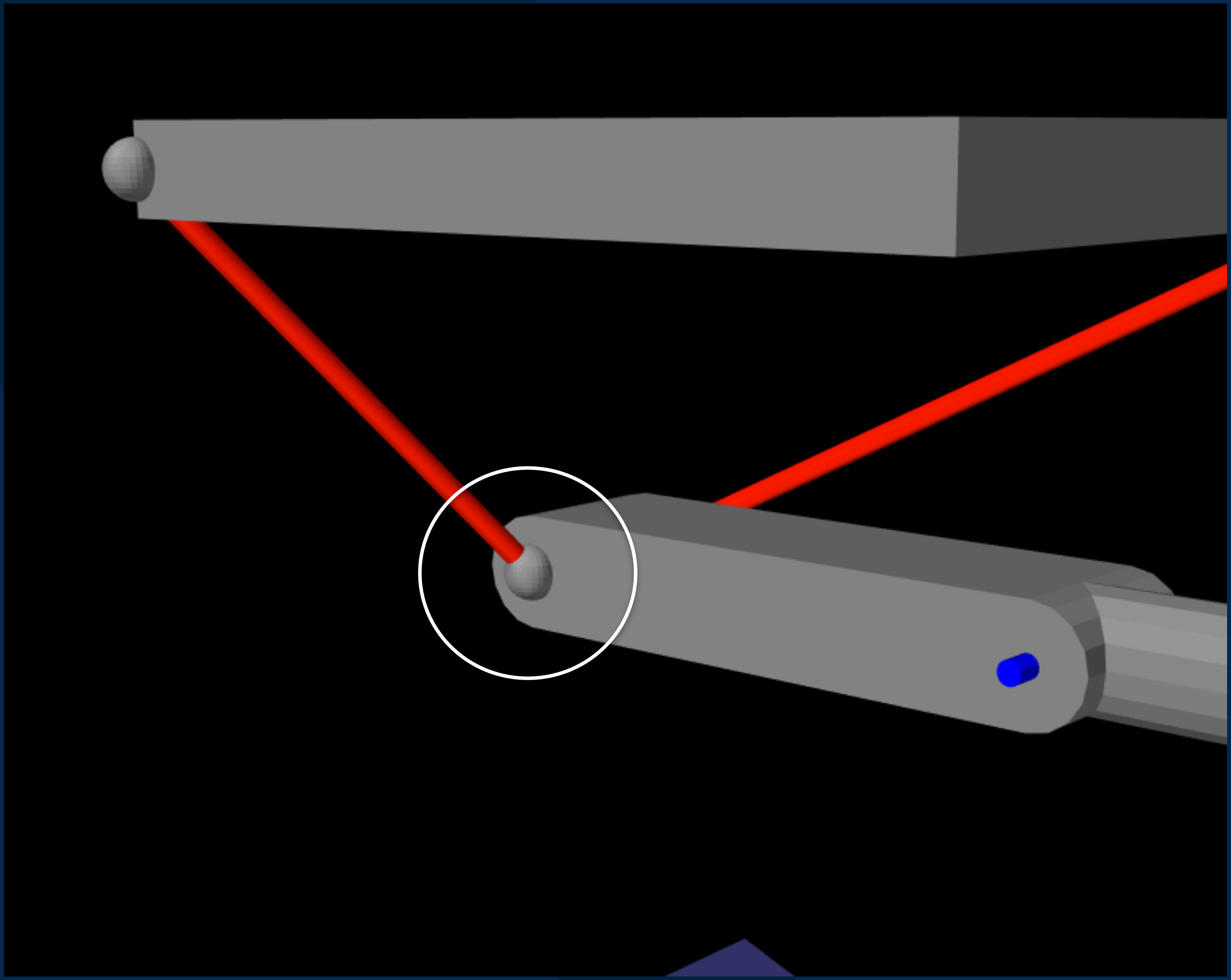


Next: rigid bodies



RigidTransform3d object describes transform X_{BA} between two frames, and also the *pose* of a rigid body

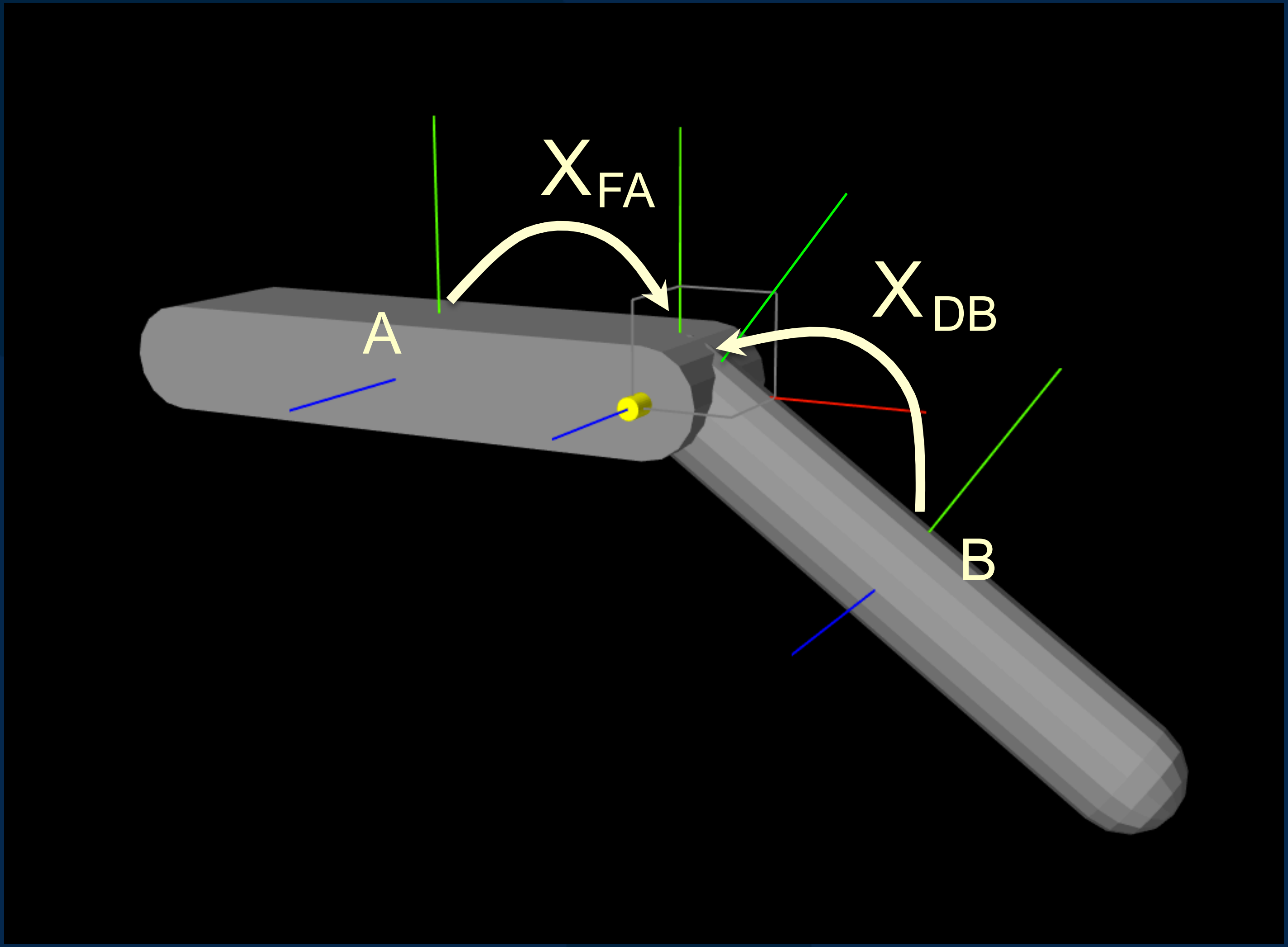
FrameMarkers used to connect to springs



artisynth.demos.tutorials.RigidBodySpring

```
MechModel mech = new MechModel ("mech");
addModel (mech);

Particle p1 = new Particle ("p1", /*mass=*/2, 0, 0, 0);

// create box and set it's pose (position/orientation):
RigidBody box =
   RigidBody.createBox ("box", 0.5, 0.3, 0.3, /*density=*/20);
box.setPose (new RigidTransform3d (0.75, 0, 0));
// create marker point and connect it to the box:
FrameMarker mkr = new FrameMarker (-0.25, 0, 0);
mkr.setFrame (box);

// create the spring:
AxialSpring spring = new AxialSpring ("spr", /*restLength=*/0);
spring.setPoints (p1, mkr);
spring.setMaterial (
   new LinearAxialMaterial (/*stiffness=*/20, /*damping=*/10));

// add components to the mech model
mech.addParticle (p1);
mech.addRigidBody (box);
mech.addFrameMarker (mkr);
```

Adding joints to rigid bodies



artisynth.demos.tutorials.RigidBodyJoint

```java
public void build (String[] args) {

   // create mech model and set it's properties
   mech = new MechModel ("mech");
   mech.setGravity (0, 0, -98);
   mech.setFrameDamping (1.0);
   mech.setRotaryDamping (4.0);
   addModel (mech);

   PolygonalMesh mesh;  // bodies will be defined using a mesh

   // create first body and set its pose
   mesh = MeshFactory.createRoundedBox (lenx1, leny1, lenz1, /*nslices=*/8);
   RigidTransform3d XMB = new RigidTransform3d (0, 0, 0, 1, 1, 1, 2*Math.PI/3);
   mesh.transform (XMB);
   body1 = RigidBody.createFromMesh ("body1", mesh, 0.2, 1.0);
   body1.setPose (new RigidTransform3d (0, 0, 1.5*lenx1, 1, 0, 0, Math.PI/2));
   body1.setDynamic (false);

```

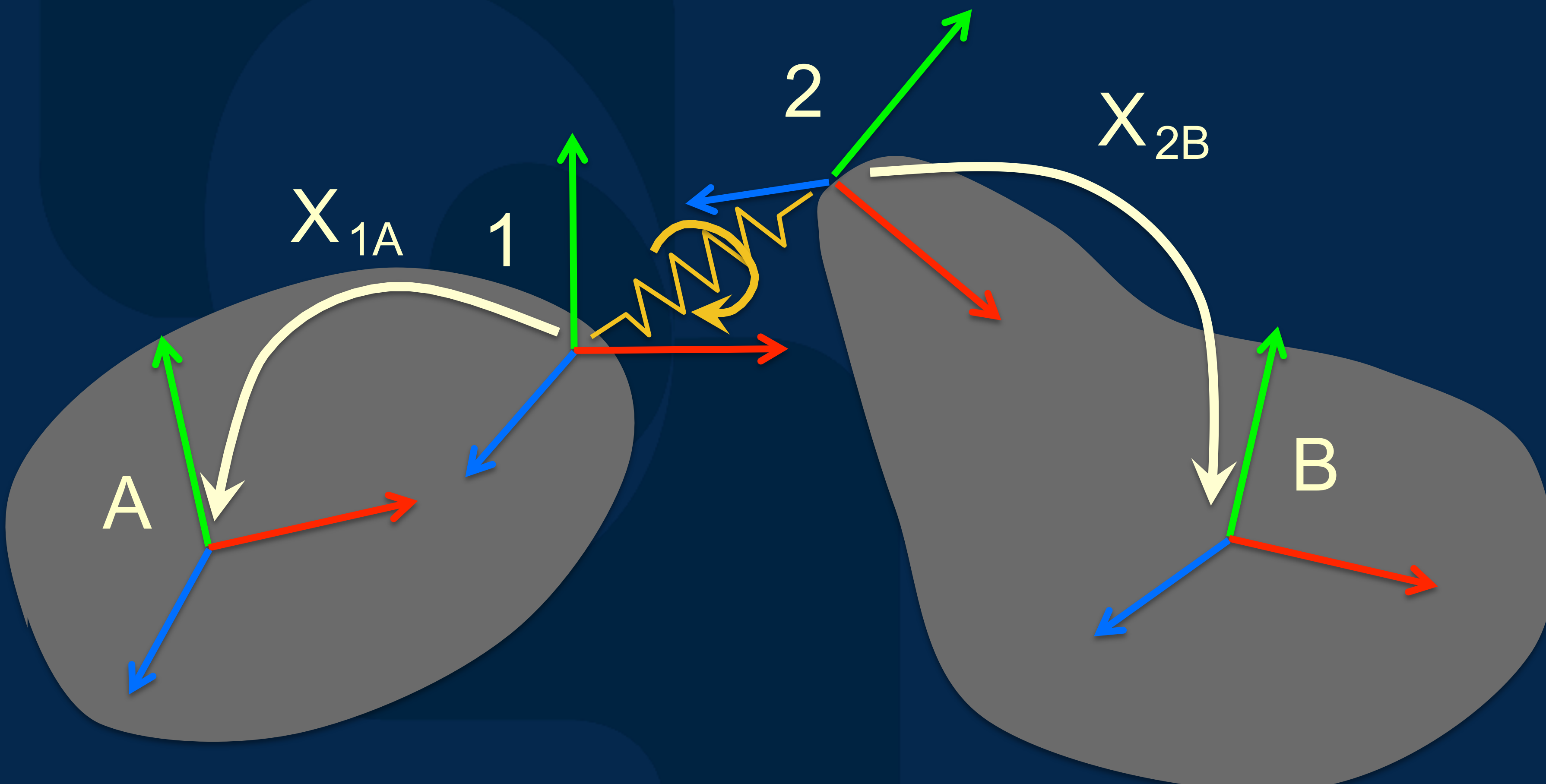
RigidBodyJoint (cont)

```java

      // create second body and set its pose
      mesh = MeshFactory.createRoundedCylinder (
         leny2/2, lenx2, /*nslices=*/16, /*nsegs=*/1, /*flatBottom=*/false);
      mesh.transform (XMB);
      body2 = RigidBody.createFromMesh ("body2", mesh, 0.2, 1.0);
      body2.setPose (new RigidTransform3d (
                        (lenx1+lenx2)/2, 0, 1.5*lenx1, 1, 0, 0, Math.PI/2));

      // create the joint
      RigidTransform3d XCA = new RigidTransform3d (-lenx2/2, 0, 0);
      RigidTransform3d XCB = new RigidTransform3d();
      XCB.mulInverseLeft (body1.getPose(), body2.getPose());
      XCB.mul (XCA);
      RevoluteJoint joint = new RevoluteJoint (body2, XCA, body1, XCB);

      // add components to the mech model
      mech.addRigidBody (body1);
      mech.addRigidBody (body2);
      mech.addRigidBodyConnector (joint);

      joint.setTheta (35);  // set joint position
```

artisynth.demos.tutorial.JointedCollide

```java
public class JointedCollide extends RigidBodyJoint {

   public void build (String[] args) {

      super.build (args);

      body1.setDynamic (true);  // allow body1 to fall freely

      // create and add the base plate
      RigidBody base = RigidBody.createBox ("base", 25, 25, 2, 0.2);
      base.setPose (new RigidTransform3d (5, 0, 0, 0, 1, 0, -Math.PI/8));
      base.setDynamic (false);
      mech.addRigidBody (base);

      // turn on collisions
      mech.setDefaultCollisionBehavior (true, 0.20);
      mech.setCollisionBehavior (body1, body2, false);
   }

}
```

Connecting rigid bodies with frame springs



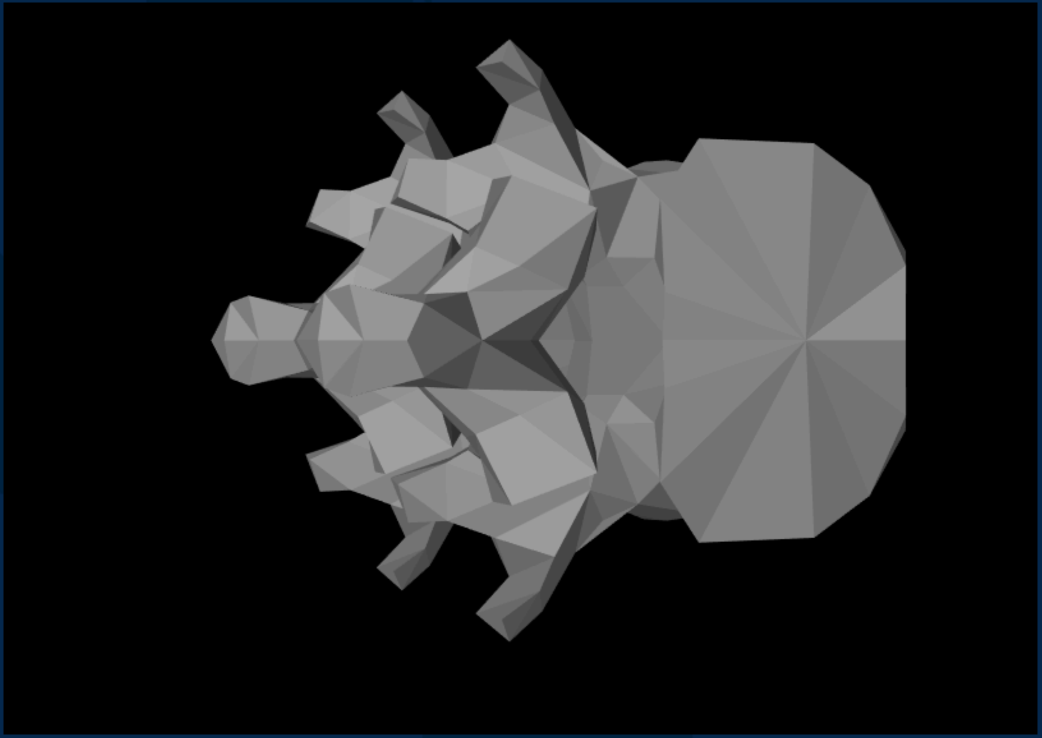
artisynth.demos.tutorial.LumbarFrameSpring

```java
public class LumbarFrameSpring extends RootModel {

   double density = 1500;

   // path from which meshes will be read
   private String geometryDir = ArtisynthPath.getSrcRelativePath (
      LumbarFrameSpring.class, "../mech/geometry/");

   // create and add a rigid body from a mesh
   public RigidBody addBone (MechModel mech, String name) throws IOException {
      PolygonalMesh mesh = new PolygonalMesh (new File (geometryDir+name+".obj"));
      RigidBody rb = RigidBody.createFromMesh (name, mesh, density, /*scale=*/1);
      mech.addRigidBody (rb);
      return rb;
   }

```

LumbarFrameSpring (cont)

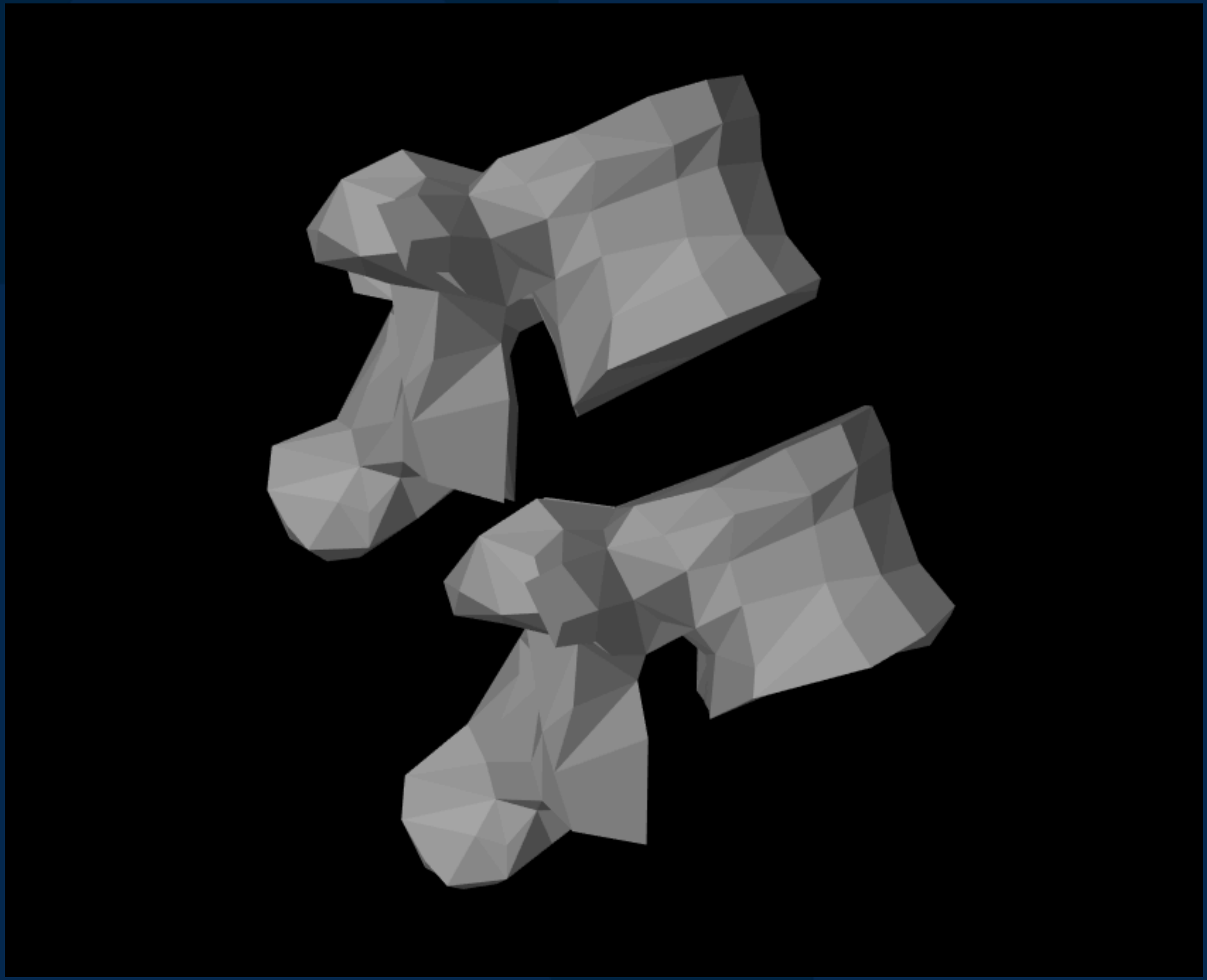
and in the build method ...

```java
// create two rigid bodies and second one to be fixed
RigidBody lumbar1 = addBone (mech, "lumbar1");
RigidBody lumbar2 = addBone (mech, "lumbar2");
lumbar1.setPose (new RigidTransform3d (-0.016, 0.039, 0));
lumbar2.setDynamic (false);
```



LumbarFrameSpring (cont)

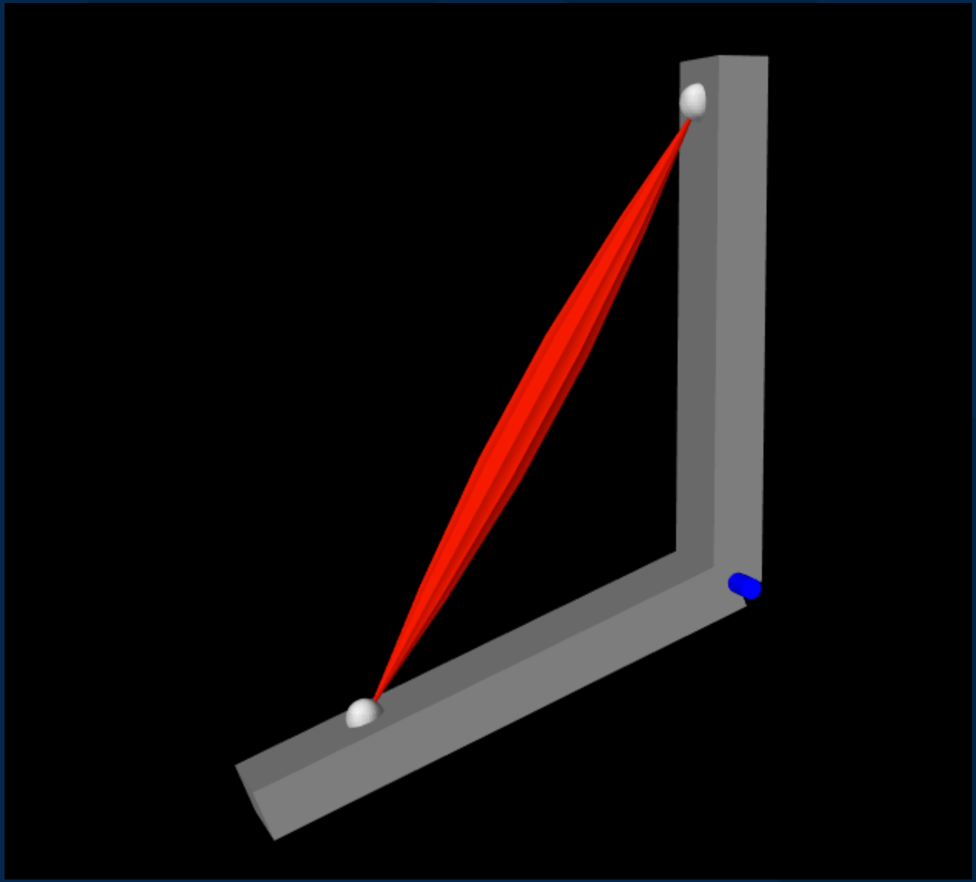
```
// flip entire mech model around
mech.transformGeometry (
   new RigidTransform3d (0, 0, 0, 0, 0, Math.toRadians (90)));
```



LumbarFrameSpring (cont)

```
// flip entire mech model around
mech.transformGeometry (
   new RigidTransform3d (0, 0, 0, 0, 0, Math.toRadians (90)));

// create and add the frame spring
FrameSpring spring = new FrameSpring (null);
spring.setMaterial (
   new LinearFrameMaterial (
      /*ktrans=*/100, /*krot=*/0.01, /*dtrans=*/0, /*drot=*/0));
RigidTransform3d X1A = new RigidTransform3d();
X1A.mulInverseLeft (lumbar2.getPose(), lumbar1.getPose());
spring.setAttachFrameA (X1A);
spring.setAttachFrameB (RigidTransform3d.IDENTITY);
mech.attachFrameSpring (lumbar2, lumbar1, spring);
```

Muscle activation



Use a Muscle:
an AxialSpring that can also apply forces

artisynth.demos.tutorial.SimpleMuscle

Same as RigidBodySpring, using Muscle instead:

```

      p1 = new Particle ("p1", /*mass=*/2, 0, 0, 0);
      // create box and set it's pose:
      box = RigidBody.createBox ("box", 0.5, 0.3, 0.3, /*density=*/20);
      box.setPose (new RigidTransform3d (1.00, 0, 0));
      // create marker point and connect it to the box:
      FrameMarker mkr = new FrameMarker (-0.25, 0, 0);
      mkr.setFrame (box);

      // create the muscle:
      muscle = new Muscle ("mus", /*restLength=*/0);
      muscle.setPoints (p1, mkr);
      muscle.setMaterial (
         new SimpleAxialMuscle (/*stiffness=*/20, /*damping=*/10, /*maxf=*/10));

```

Questions?