Overview

- Some background ideas
- Installation and a first example
- Device descriptions and basic analysis
- Some ongoing work
- Break

What should you get out?

- Understanding of what SUGAR does
- Ability to use and get help on SUGAR commands
- Understanding of how to build netlists
- Ability to do basic analyses

Simulation: How?

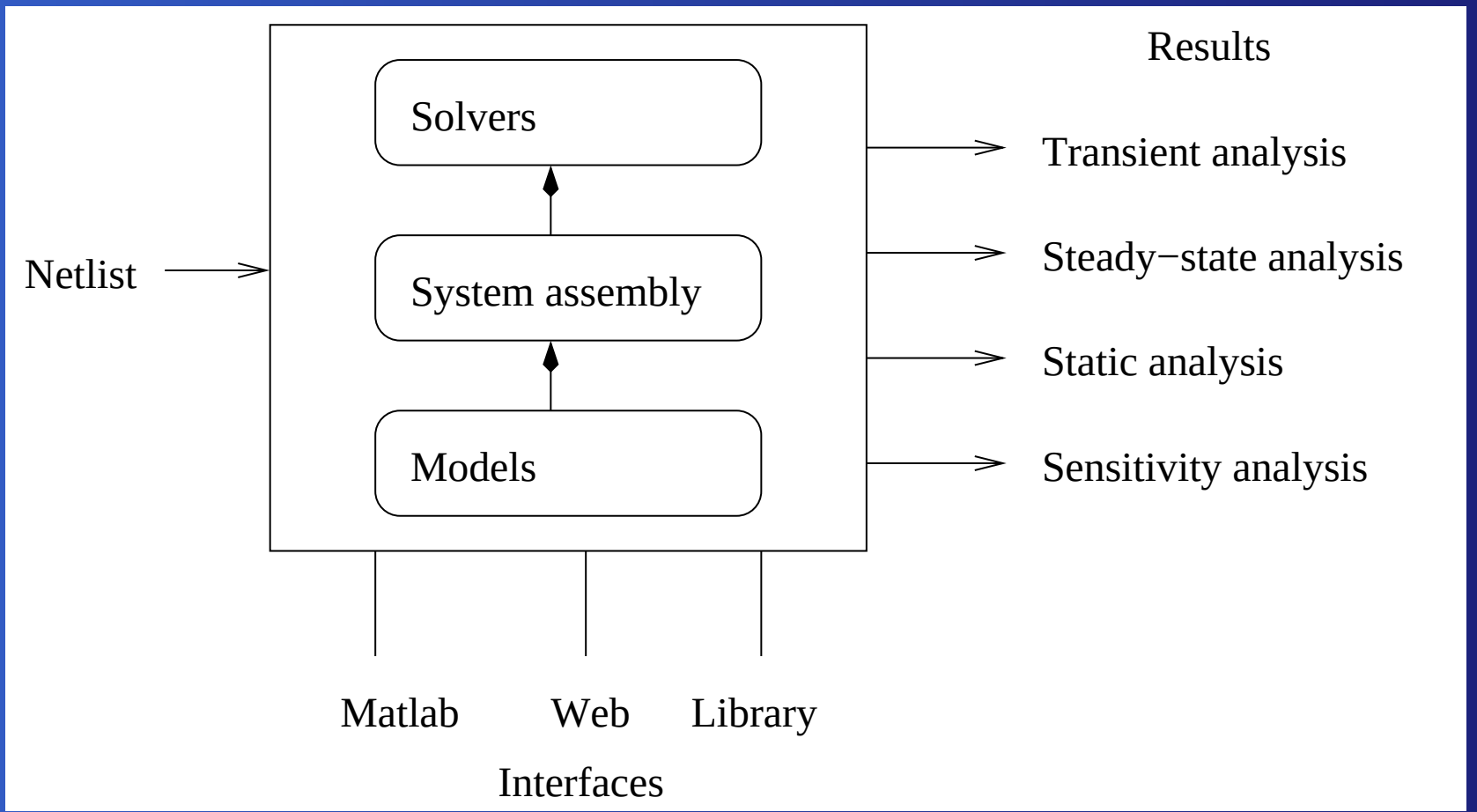
- Solve continuum equations (finite elements, finite differences, boundary elements)
- Solve simplified equations of beam and plate theory (finite elements, etc)
- Solve network equations (e.g. modified nodal analysis)
- These approaches are not mutually exclusive!

Standard discrete system

Finite elements, finite differences, and network equations all have similar structure. Standard steps of discretization (courtesy Zienkiewicz and Taylor):

- Write local equations governing components
- Assemble global equations (by addition)
- Apply boundary conditions
- Solve global equations and process output

SUGAR architecture



Starting SUGAR

- You need Matlab for SUGAR 2.0!
- Unpack the source directory
- Add the analysis and model subdirectories to the Matlab path (use addpath)
- Try running `demo_cantilever` from the demo directory

Cantilever walk-through

```
net = cho_load('cantilever.net');
cho_display(net);
dq = cho_dc(net);
cho_display(net, dq);
dy = dqval(net, dq, 'tip', 'y')
```

- Load and display device description
- Analyze and display static displacement
- Get y-displacement of tip

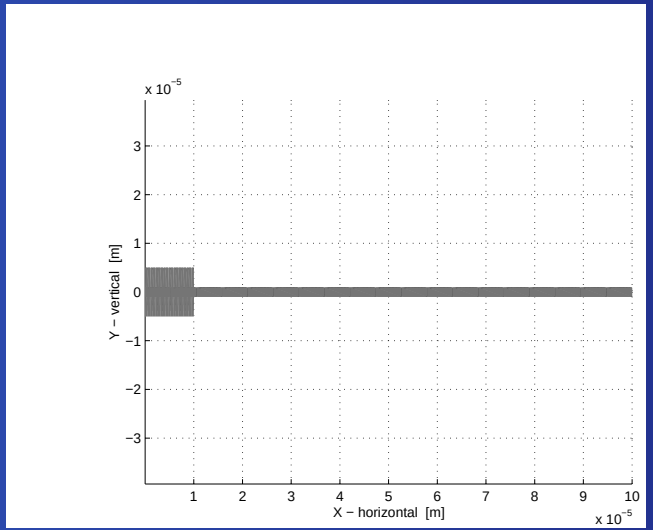
Netlist notes

- Devices are described by text *netlists*
- Can be *parameterized* and *heirarchical*
- Lists of elements connected together at nodes
- Similar to SPICE or finite element netlists (but with nicer syntax)

Basic lexical and syntax notes

- Can include files via a `uses` statement
- Use square brackets as basic punctuation
- Numbers can have metric suffixes (e.g. `2u` for two micrometers)
- Comments begin with %, extend to end of line
- Can form simple numerical expressions, Matlab-like syntax for operations
- Can define variables (e.g. `A = 2u*2u`)

Netlist syntax




```
uses mumps.net

anchor  p1 [substrate]     [l=10u w=10u]
beam3d  p1 [substrate tip] [l=100u w=2u h=2u]
f3d     *   [tip]          [F=2u oz=pi/2]
```

Node positioning

Can specify rotation angles (rotate about x, z, then y axes):

```
beam3d p1 [tip top]
          [w=2u l=100u oz=pi/2]
```

OR can specify relative node positions:

```
pos     *  [tip top] [y=100u]
beam3d p1 [tip top] [w=2u]
```

Process information

- *Process* info describe material parameters, etc.
- Canonical example is `mumps.net`

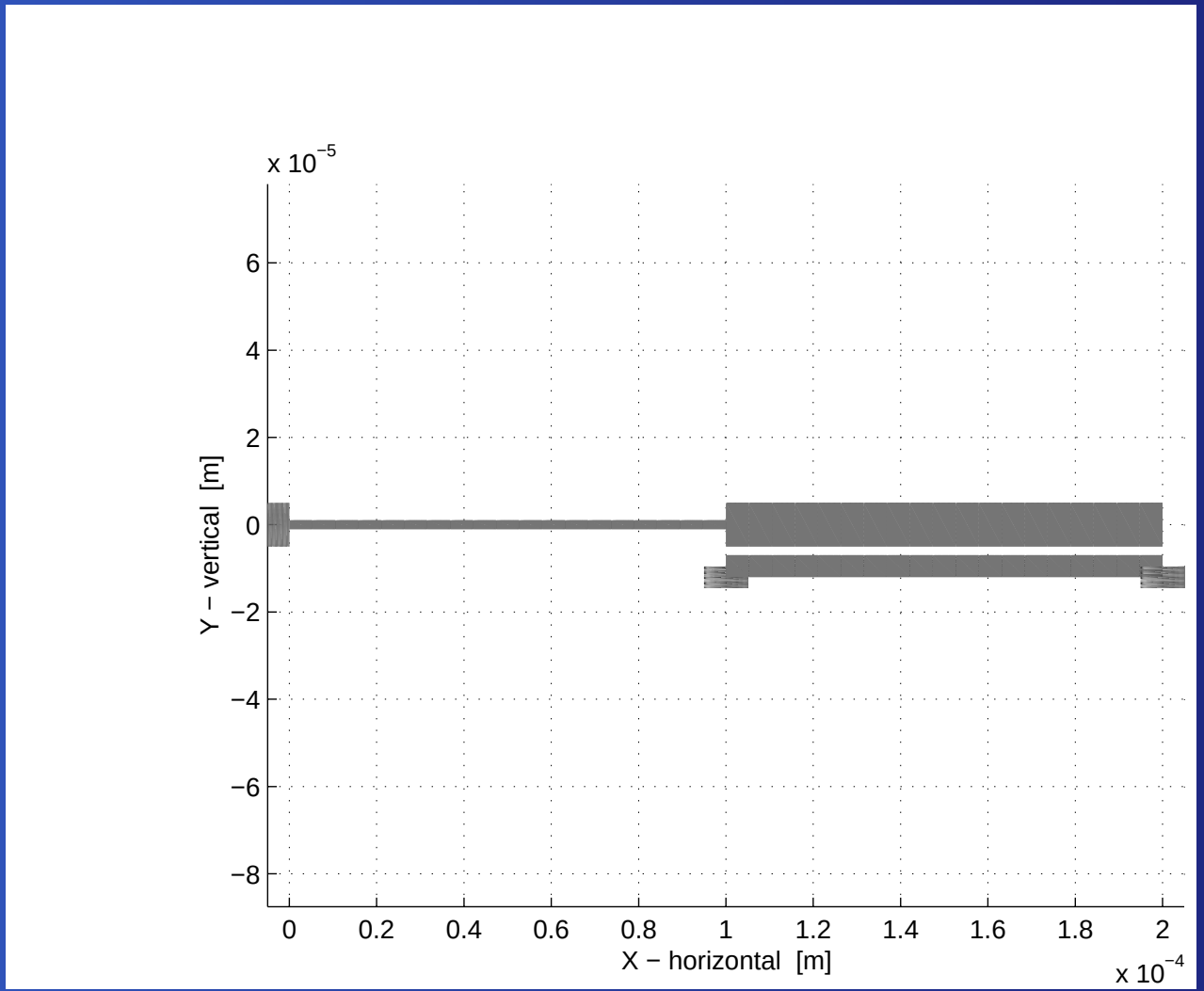
```
process poly = [
    Poisson = 0.3
    ...
]
```

Process inheritance

Idea: type in generic process information once, override parameters to get specific layers

```
process p1 : poly = [
    h = 2u
]
```

Example: Gap-closing actuator



Gap actuator netlist

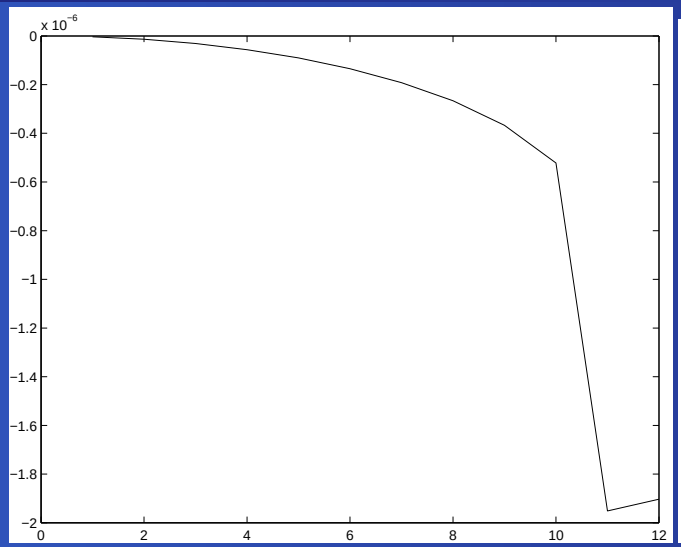
```
uses mumps.net
uses stdlib.net

Vsrc      *   [A f] [V=10]
eground   *   [f]   []
anchor    p1  [A]   [l=5u w=10u  oz=deg(180)]
beam2de   p1  [A b] [l=100u w=2u h=2u R=100]
gap2de    p1  [b c D E] [l=100u w1=10u w2=5u
                         gap=2u R1=100 R2=100]
eground   *   [D]   []
anchor    p1  [D]   [l=5u w=10u oz=-deg(90)]
anchor    p1  [E]   [l=5u w=10u oz=-deg(90)]
eground   *   [E]   []
```

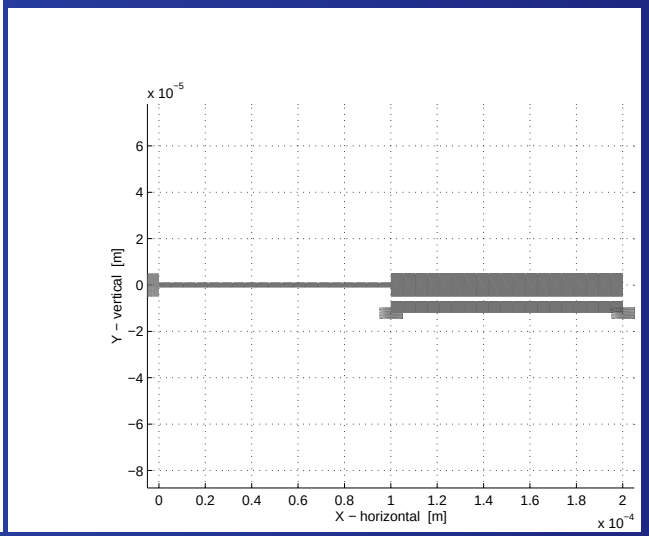
Adding parameterization

```
param V = 10
Vsrc  *  [A f] [V=V]
```

- V can be set by user at load time
- If no value explicitly provided, use 10V

Using the Matlab interface






```
dq = [];
for k=1:12
   param.V = k;
   net = cho_load('beamgap2b.net', param);
   dq = cho_dc(net, dq);
   tip(k) = cho_dq_view(dq, net, 'c', 'y');
end
```

Subnets and heirarchical design

- *Subnets* are parameterized components
- Subnet calls look like built-in model calls
- Matches design heirarchies
- Can put commonly-used subnets in a library

Subnet parameters

```
subnet foo [a b] [l=* w=5u h=?]
[ ... ]

foo p1 [q r] [l=100u]
```

- a and b are local names for interface nodes
- `l` is a required parameter
- `w` is required, but has a default value
- `h` is an optional parameter

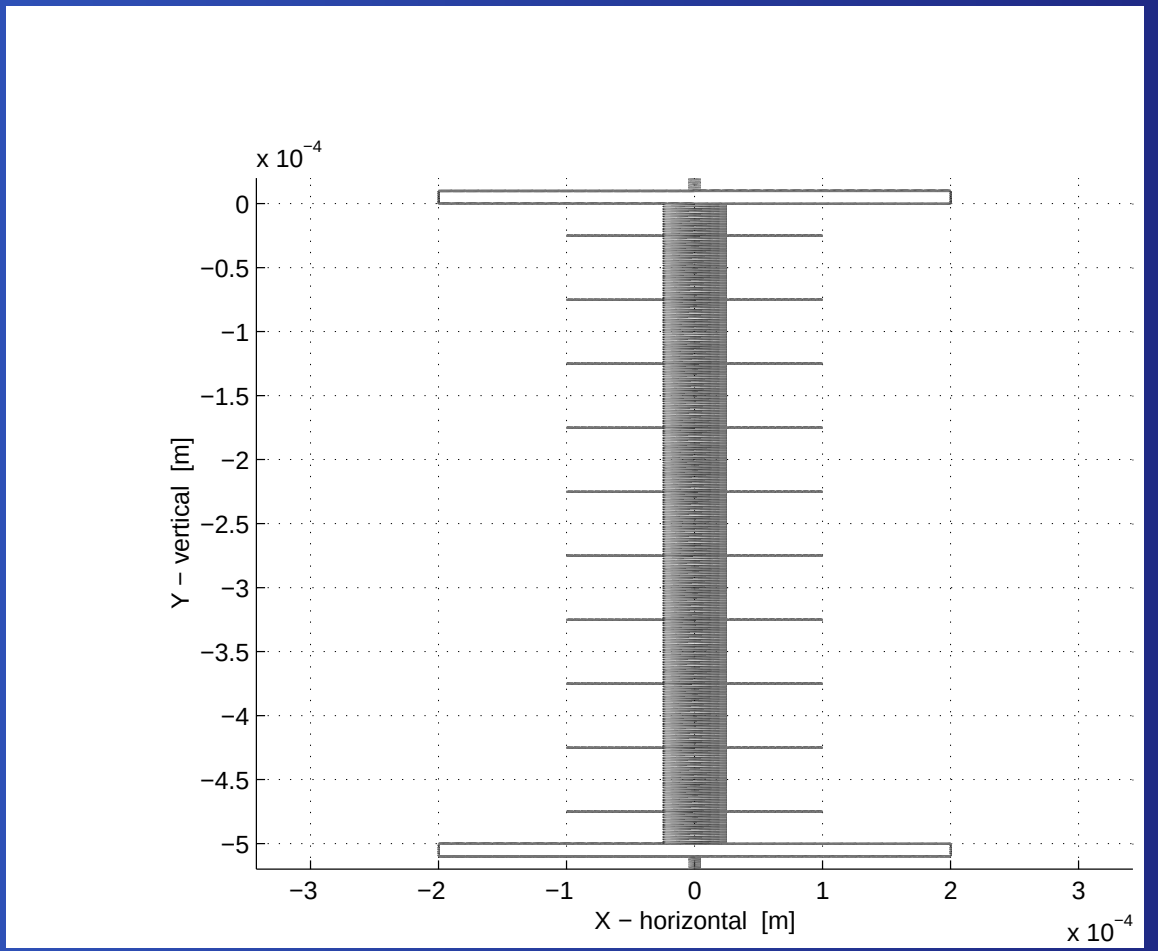
Process arguments

- The `parent` process is the process passed to a subnet instance
- Parameters without defaults or explicitly assigned values will try to use values from `parent`
- The `parent` can be used as an argument to element lines inside the subnet

Nested coordinate systems

- The `ox`, `oy`, and `oz` subnet parameters are reserved
- Used to determine a local coordinate system for the subnet
- Nested subnets result in multiply nested coordinate systems
- Will have a simple example shortly

Example: simplified ADXL-50



Parameterization

```
param nfingers = 10
```

- `nfingers` can be set by user at load time
- If no value explicitly provided, use ten fingers

Subnet example

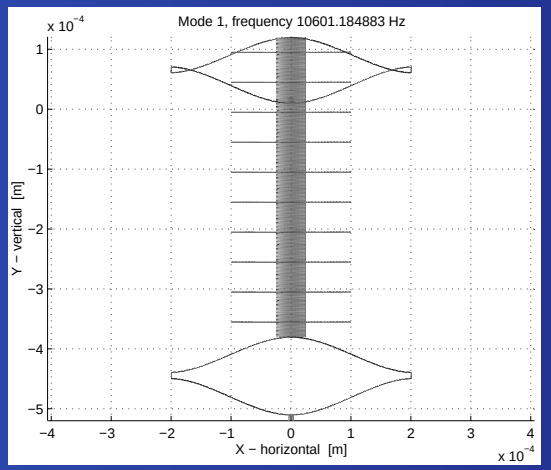
```
subnet XMass [A B] [finger_len=*]
[
  b1 beam3d parent [A  b1] [l=25u         w=50u
                            h=6u oz=-deg(90)]
  b2 beam3d parent [b1 B ] [l=25u         w=50u
                            h=6u oz=-deg(90)]
  b3 beam3d parent [b1 b2] [l=finger_len w=2u
                            h=6u oz= deg(0) ]
  b4 beam3d parent [b1 b3] [l=finger_len w=2u
                            h=6u oz=deg(180)]
]
```

Arrays

```
XSusp p1 [c(1)] [susp_len=200u]
for k=1:nfingers [
  mass(k) XMass p1 [c(k) c(k+1)] [finger_len=100u]
]
XSusp p1 [c(11)] [susp_len=200u oz=pi]
```

- Suspensions at either end (note rotated subnet!)
- `nfingers` comb-finger units as defined above

Modes of ADXL-50 model



(Displacements are exaggerated)

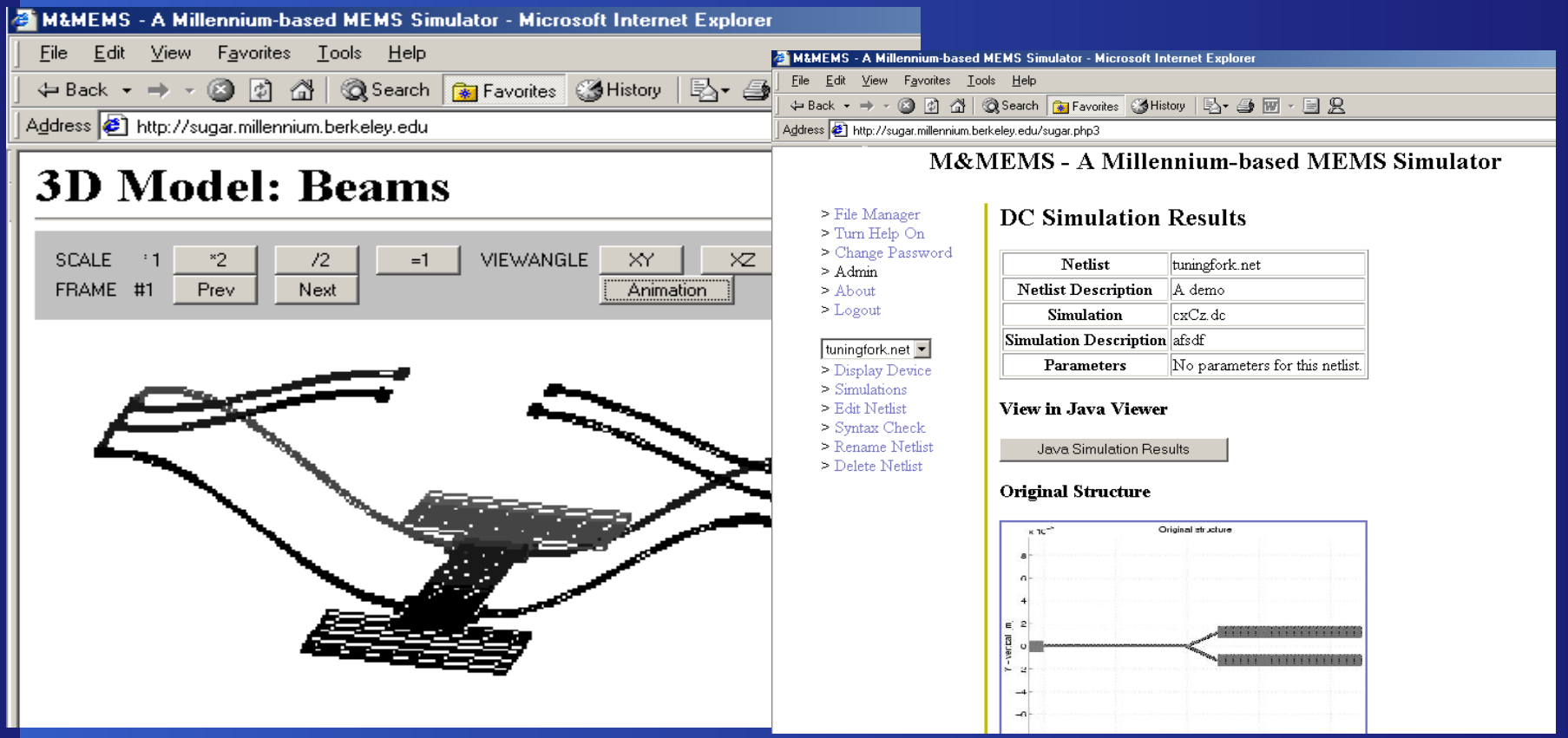
```
net = cho_load('adxl.net');
[f,e,dq] = cho_mode(net);
cho_modeshape(net, f,e,dq, 1);
```

SUGAR 3.0

- SUGAR 2.0: Matlab core, C add-ons
- SUGAR 3.0: C core, Matlab interfaces
- Release target: some time this month?
- Web service using 3.0 should be more robust
- Can use SUGAR 3.0 even without Matlab
- Netlist language has same spirit, slightly different syntax

M&MEMS: SUGAR on the Web

http://sugar.millennium.berkeley.edu/



And more!

- Models: more and better
- Improved numerics and additional analyses
- Feedback from measured data
- SUGAR as a component in design optimizer