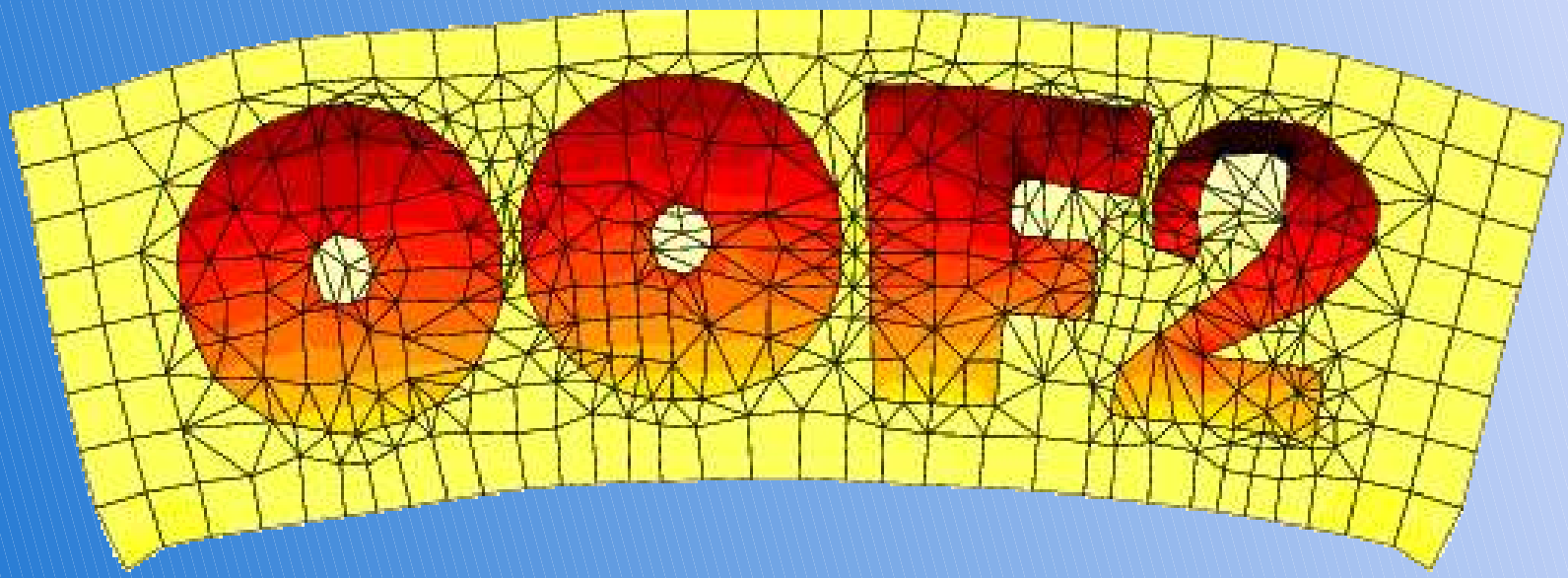


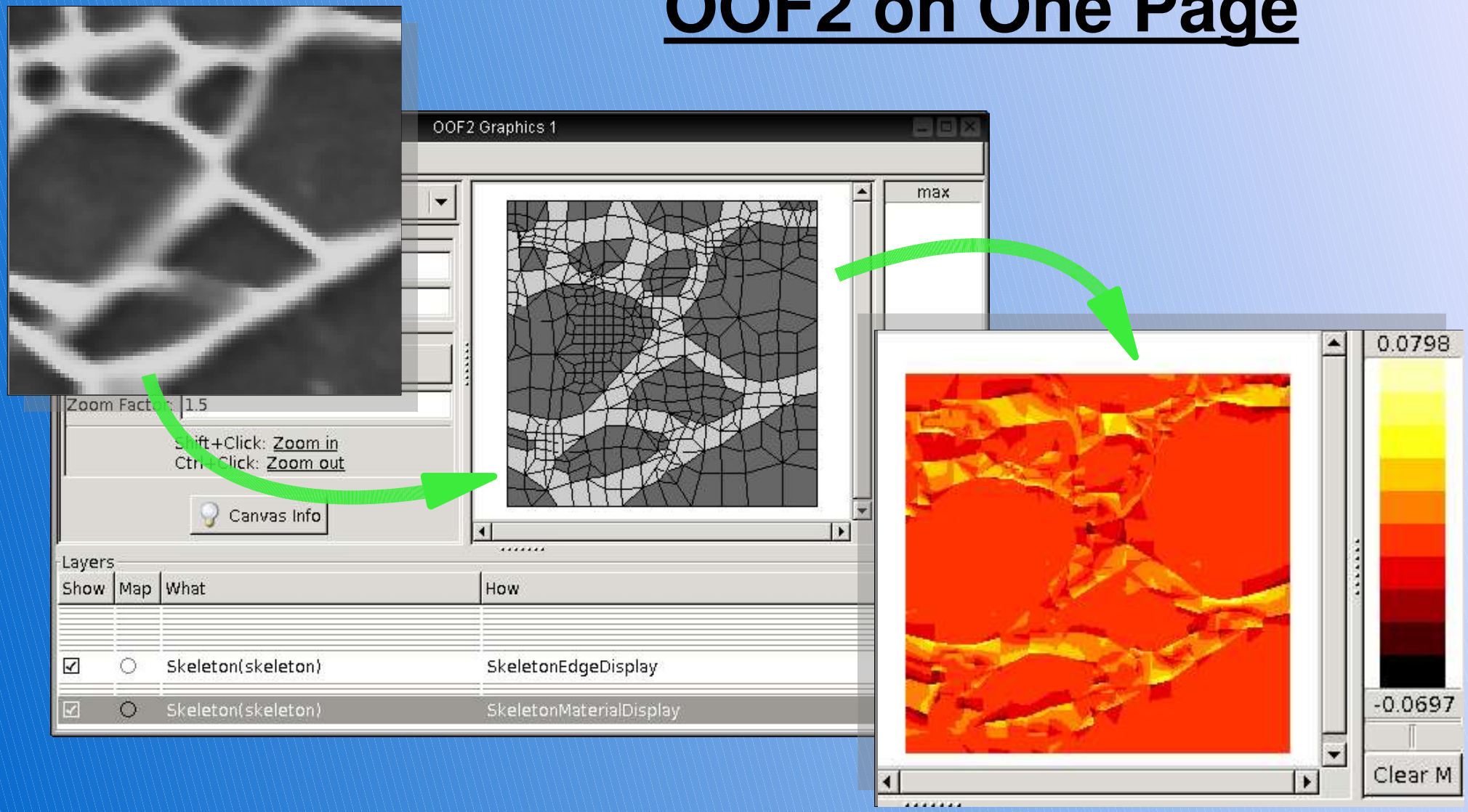
Object-Oriented Finite Elements

OOF Workshop overview, August 24, 2006

NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Drexel
UNIVERSITY

OOF2 on One Page



Outline

- Motivation for the OOF project
- OOF1 vs OOF2
- Scope of OOF2
- Conceptual organization of OOF2
- Solution operations
- Software architecture

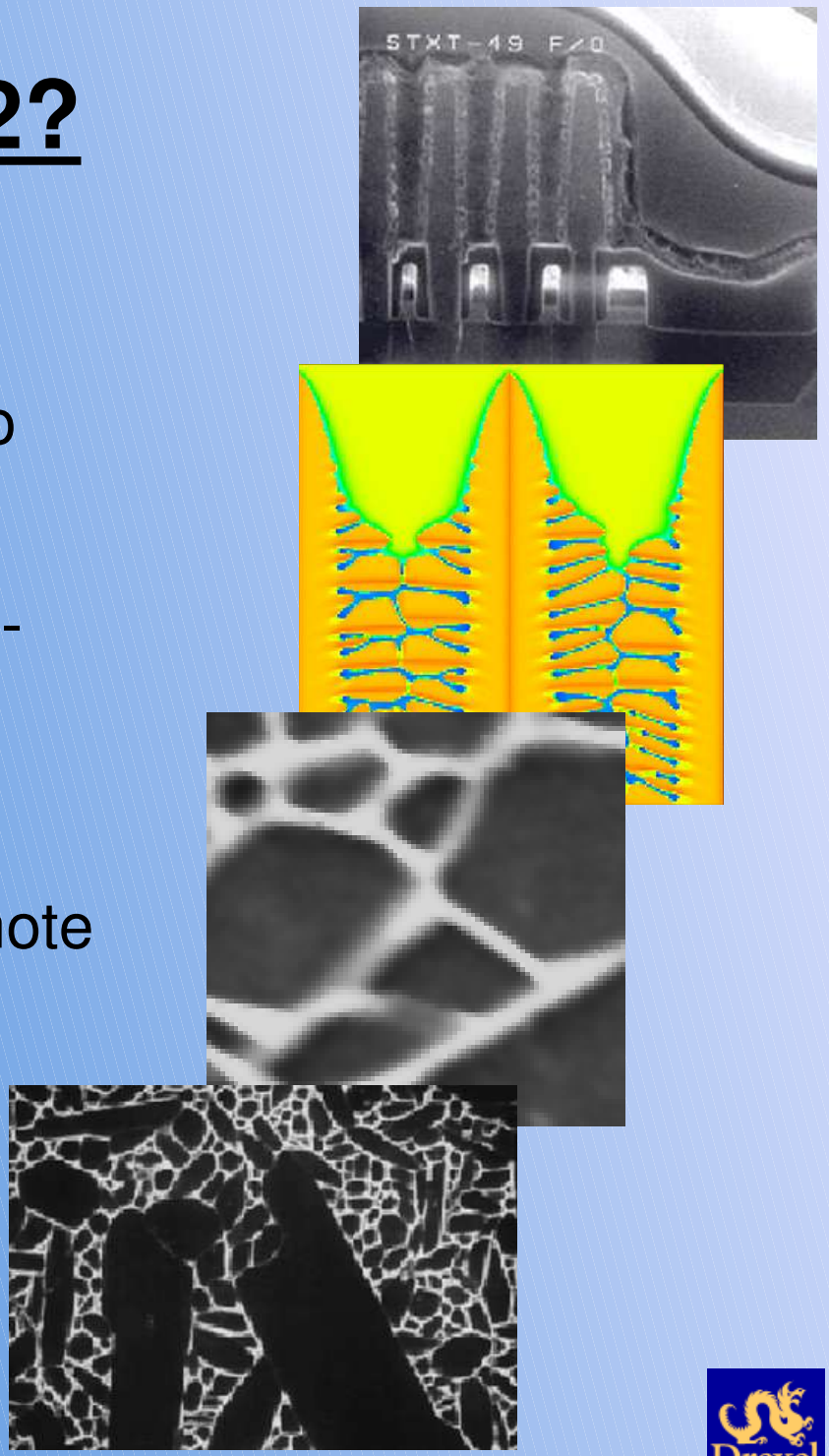
Why OOF2?

- Commercial FEM packages exist, but are best suited to domains with clean mathematical descriptions.
- Materials scientists ask, “How does microstructure affect aggregate properties?”
  - Irregular structure is *important*
- Possible but tedious to construct a microstructural FEM mesh in a commercial code

Why OOF2?

OOF2 offers:

- *Rapid* construction of meshes adapted to irregular microstructural geometries
- Constitutive rules expressed in materials-science terms
- User extensibilty of constitutive rules
- User-friendly GUI, command-line, or remote operating modes
- Convenient parametric variations
  - "virtual experiments"



Why OOF2?

Why Government?

- Multidisciplinary broad scope
  - Suitable for large institution
- Multi-year development time
- Absence of specific-material focus
- Niche audience

OOF1 vs OOF2

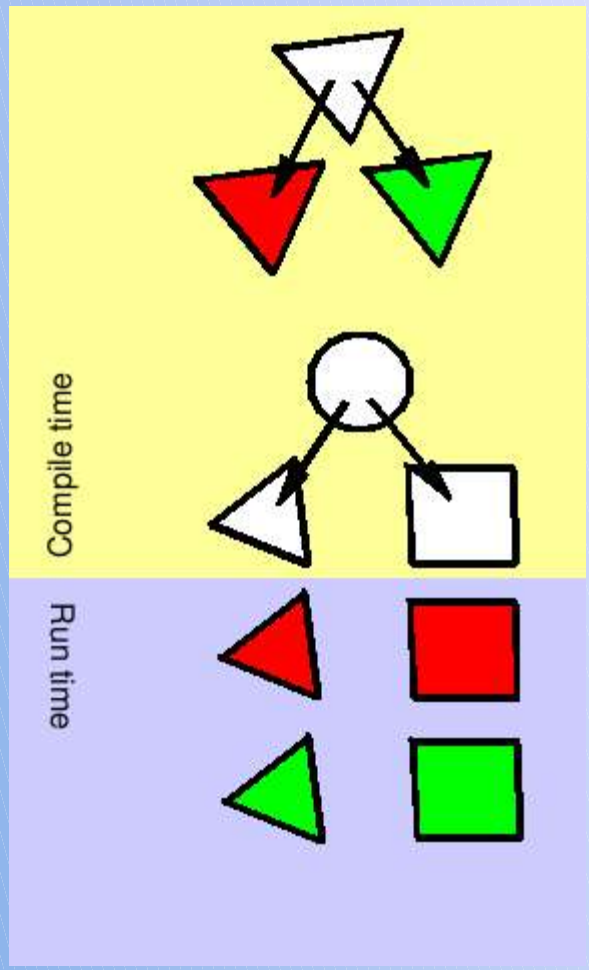
OOF1 was the first cut at building an adaptable, extensible, materials-science oriented FEM tool

- Proved numerous mesh-adapting tools for triangular meshes
- Proved large *potential* scope through customized versions constructed by Edwin García
  - Diffusion, piezoelectricity, electrochemistry
- Proved trickiness of extensible software

OOF1 vs OOF2

Materials

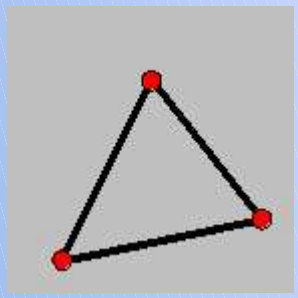
- OOF1 materials self-contained, “atomic”
  - Coupled tightly to element geometry
  - Modification requires recompilation
- OOF2 materials built up from “property” objects
  - Independent of element geometry
  - Modifiable from the GUI



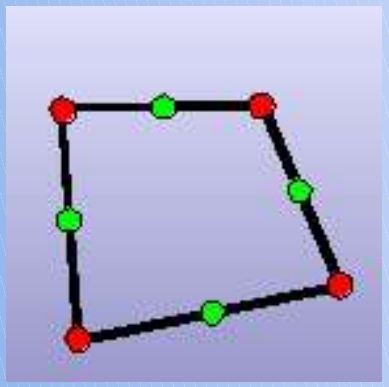
OOF1 vs OOF2

FEM capabilities

- OOF1 contains 1st order triangular elements
  - Coupled to materials



- OOF2 contains numerous element types
  - 1st and 2nd order interpolation
  - Triangles and quadrilaterals
  - Isoparametric and subparametric



OOF1 vs OOF2

Logistics

- OOF1 is two programs
  - ppm2oof constructs meshes
  - oof solves them
- OOF1 can only read ppm-format image files

- OOF2 is one program, but can be loaded modularly
- Can read many different image formats
  - ppm, jpg, tiff, png...

OOF1 vs OOF2

Structure

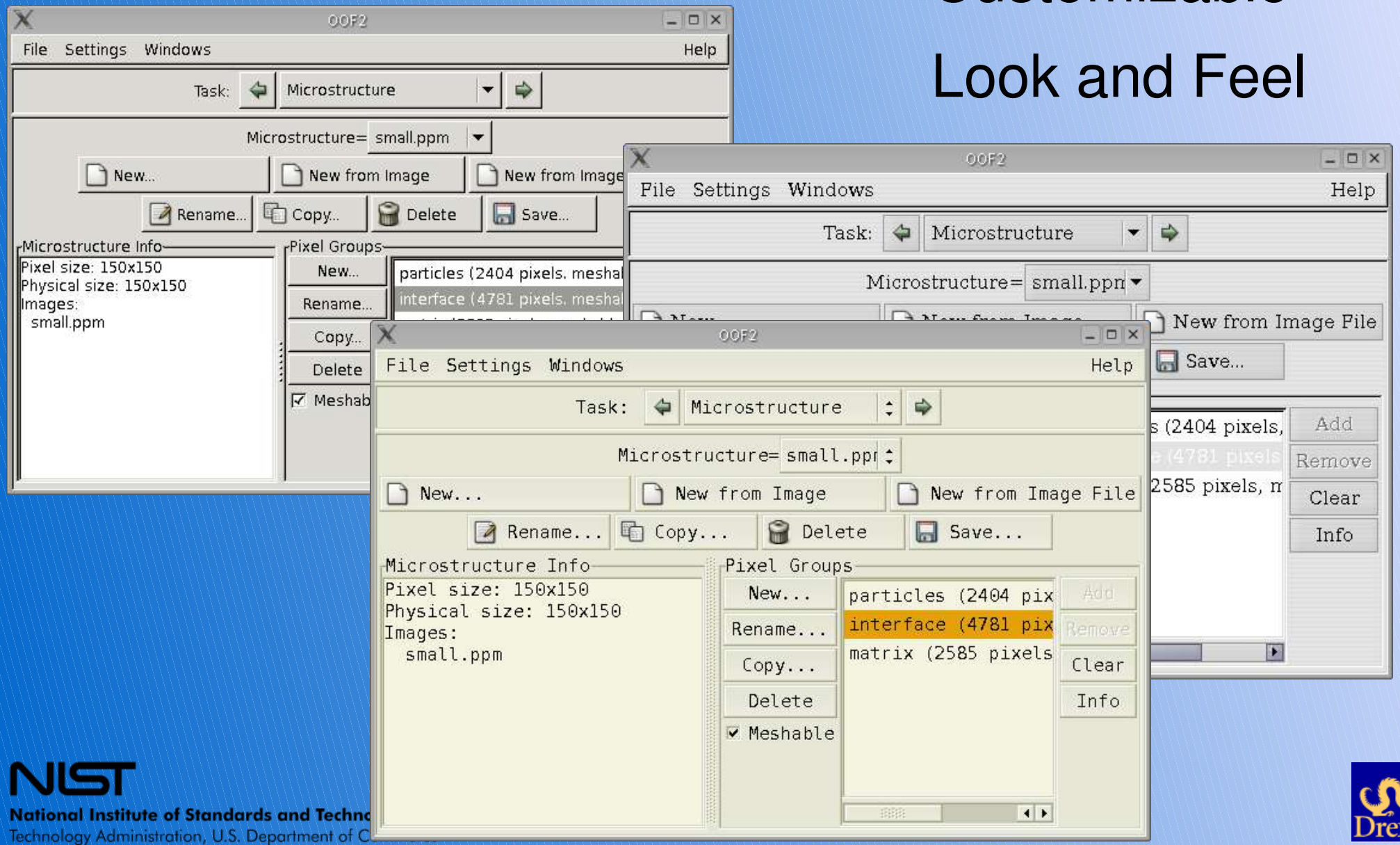
- OOF1 is written entirely in C++
  - discourages experimentation
- OOF1 depends on the XFORMS library
  - Obsolete?
  - UNIX-only

- OOF2 is written in a combination of Python and C++
  - High execution speed where required
  - Flexible high-level Python scripting, glue
    - Loops, conditionals, etc.
  - Python is popular, free, multiplatform (Windows!)
  - Uses gtk2 and pygtk toolkits, also free and multiplatform

OOF1 vs OOF2

Customizable

Look and Feel



OOF1 vs OOF2

Extensibility

- Most aspects of OOF2 are user-extensible (Major point!)
  - Fields
  - Equations
  - Properties
  - Output quantities
  - Finite-element types
  - Equation types

Extension strategy is itself flexible

- Build-time
- Run-time

OOF1 vs OOF2

The Future

- OOF2 will be the basis for all future improvements

Scope

A Brief OOF Glossary

- **Field:** A degree of freedom defined at the nodes, for which you are solving, e.g. displacement, temperature
- **Flux:** A quantity whose distribution determines the equilibrium of the system, e.g. stress, heat flow
- **Property:** A mapping between fields or derivatives of fields, and fluxes, controlled by a **modulus**, e.g. $\sigma_{ij} = C_{ijkl}\varepsilon_{kl}$
- **Equation:** A criterion on the fluxes which, when satisfied everywhere, means the system is in equilibrium, e.g. $\nabla_j \cdot \sigma_{ij} = f_i$
- **Force:** A quantity which acts as a source or sink of flux, often conjugate to a degree of freedom

Present Scope

- Divergence of flux equals source
- Flux equals coefficient times field derivatives

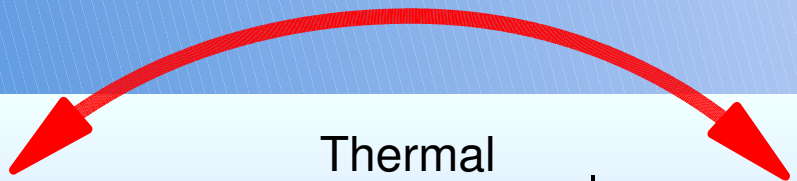
$$\nabla \cdot \sigma = f \qquad \sigma = \sum_i k_i \nabla \phi$$

	Elasticity	Thermal Conductivity	Electrostatics
flux, σ	Stress	Heat flux	Polarization
modulus, k	C_{ijkl}	κ_{ij}	ε_{ij}
field, φ	Displacement	Temperature	Voltage
force, f	Force	Heating rate	Charge

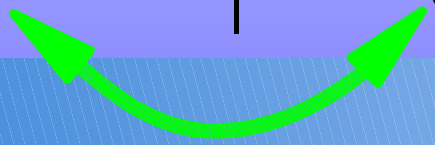
Present Scope

Also includes couplings

- **Piezoelectricity**
- **Thermal Expansion**



	Elasticity	Thermal Conductivity	Electrostatics
flux, σ	Stress	Heat flux	Polarization
modulus, k	C_{ijkl}	κ_{ij}	ε_{ij}
field, φ	Displacement	Temperature	Voltage
force, f	Force	Heating rate	Charge



Future Scope

Near term:

- Time-dependent problems
  - 1st and 2nd order in time
- Inequality constraints
- Parallelization for large systems
- Periodic boundary conditions
- 3D prototype (command-line mode)
- Application-driven development

All of continuum solid mechanics (creep, plasticity, etc.)

Longer term:

- New equation classes
  - Helmholtz, Wave
- Interactive 3D with GUI

Future Scope

User-determined:

- Library of values for existing properties
- Code for new properties
- New materials
- Application-driven extensions

Conceptual Organization

Material • Collection of properties

Image • Associates colors with pixels

Microstructure • Associates materials with pixels

• Contains pixel groups, selection, active area

Skeleton • Geometry of mesh

• Element, segment, node selections, groups

Mesh • Equations, fields, boundary conditions

• Interpolation, stiffness matrix construction

Solution data

Conceptual Organization

Materials:

- Made up of properties

Properties:

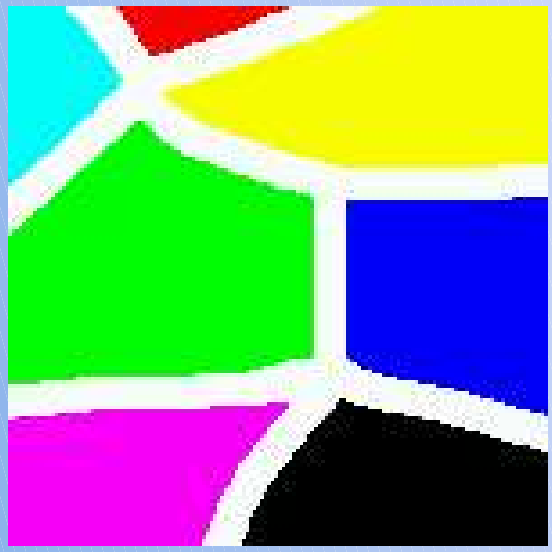
- Constitutive rules mapping fields to fluxes (mostly)
- User-quantified at run-time
  - Several different formats
- User-extensible

e.g. Elastic stiffness, thermal conductivity, orientation, color

Conceptual Organization

Image:

- Chronologically first
- Originates outside of OOF2
- Approximation to microstructure



Basic image task is to segment the image into material regions

Conceptual Organization

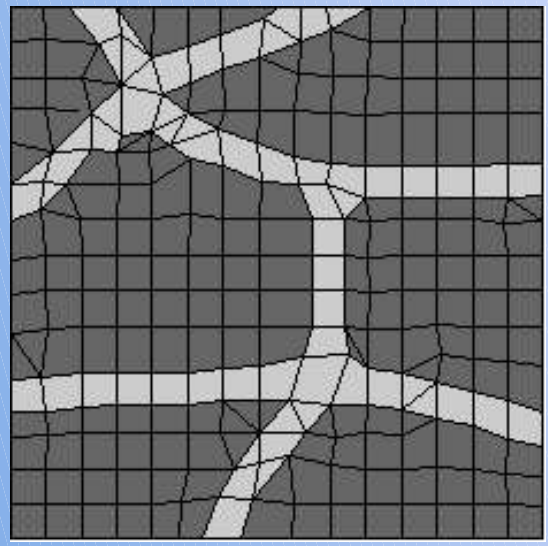
Microstructure:

- Highest level container object
- Analog of “document” in e.g. presentation program
- Maps pixels to pixel attributes, including materials
- Hosts pixel selections and groups
- Can be displayed using material color

Conceptual Organization

Skeleton:

- Contains proto-elements, segments, nodes
  - Nodes only at corners
- Has selection state
- Has numerous tools to adapt it to microstructure
- Maintains history – “undo” capability
  - For both selections and modifications

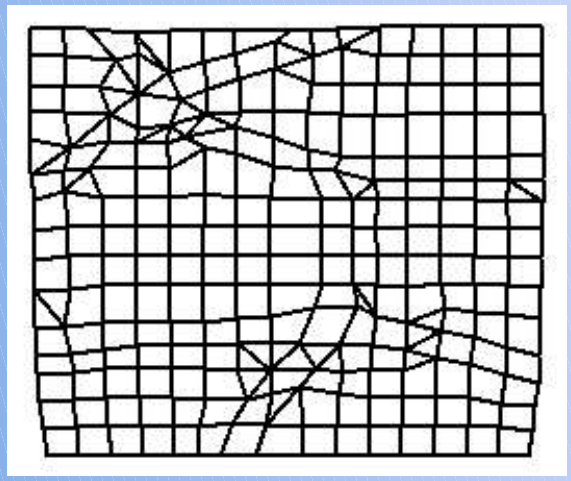


Basic skeleton task is to adapt FE geometry to microstructural geometry

Conceptual Organization

Mesh:

- Hosts the physics of the problem
  - Equations, fields
- Has boundary conditions
- Builds the master stiffness matrix
  - Calls properties to compute flux contributions

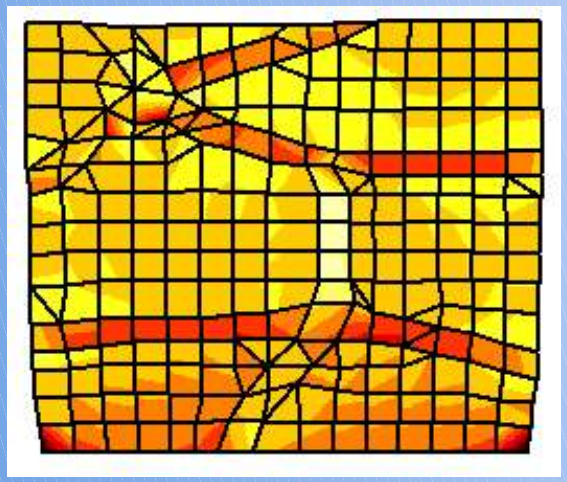


Basic mesh task is to implement the physics of the problem under study

Conceptual Organization

Solution:

- Point of the exercise!
- Numerous tools for examining OOF data
  - Fields, fluxes
  - Pointwise, statistical
  - Imaging
    - Contour maps



Solution Operations

Boundary Conditions

- **Dirichlet**
  - Require a field to have particular values along a boundary
- **Neumann**
  - Require a flux's normal component to have particular values along a boundary
- **Floating**
  - Require a field to have a particular profile, allow profile to relax
- **Force**
  - Apply specified point forces at particular nodes

Future expansion plans include periodic and “mean field” BCs

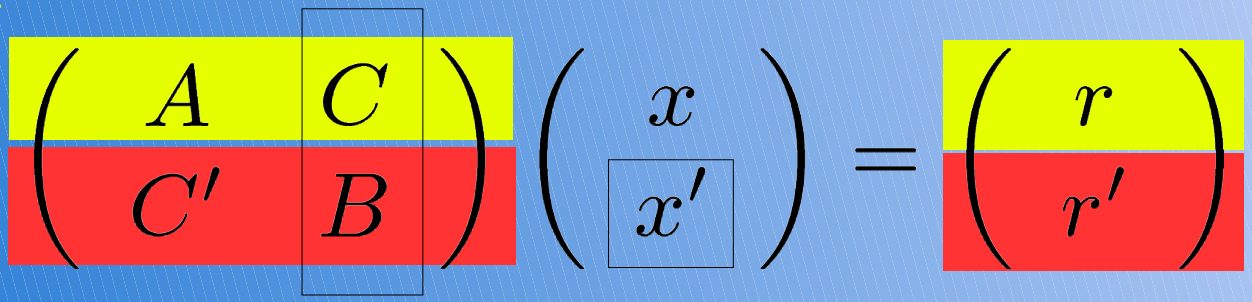
Solution Operations

Boundary Conditions

- Dirichlet and Float conditions divide up the matrix
- Neumann and Force conditions contribute to rhs

Independent equations

Dependent equations



Fixed degrees of freedom

$$A \cdot x = r - C \cdot x'$$

Solution Operations

Symmetrization

Symmetric matrices converge faster!

- Impossible if properties are not symmetric
- Possible but not guaranteed if they are
  - Collect “conjugacy” data from properties
  - Re-order equations to enforce symmetry
  - Detect this, use symmetric solvers (e.g. CG)

Solution Operations

Plane Flux (e.g. plane stress)

- Implemented as a set of auxiliary nodal equations
- Satisfied on average over a mesh shape function
- Not satisfied pointwise everywhere on the mesh
  - Prior knowledge of constitutive rule required for this

Solution Operations

Solve the linear system:

- Iterative sparse solvers
  - Conjugate Gradient, GMRES, ...
- Choice of preconditioners
  - ILU, IC, ...
- Place solution field values in mesh object
- Repeat high-level iteration (nonlinear case)
- Iterate forward (time-dependent case)

Solution Operations

Solvers:

- Currently using SparseLib++ solver set
  - Good experience with preconditioners, solvers
- Promising results with PETSc parallel solvers
- “Hard part” of parallelization

Solution Operations

Analysis

Several built-in analytical tools

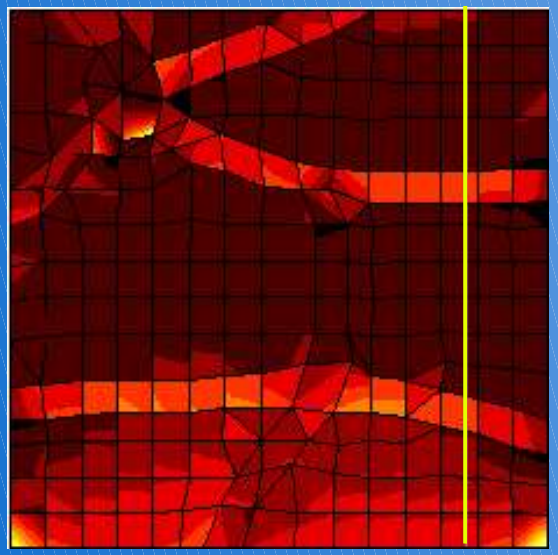
- Choice of domains
  - Element sets, rectangles, cross-sectional cuts
- Point-wise data output
  - Scalars (e.g. stress invariants)
  - Aggregates (e.g. all stress components)
- Statistical output
  - Average, standard deviation

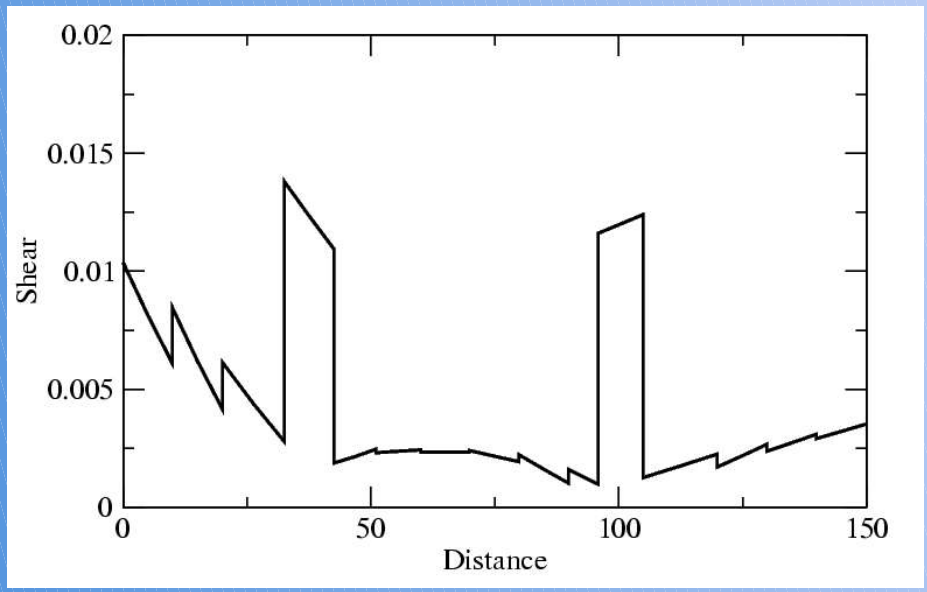
Solution Operations

Analysis

E.g. 2nd tensor invariant of shear along vertical cross section

- Uses element-aware sampling, discontinuties clear





Architecture

- Written in a combination of Python and C++
- Does not use NumPy, SciPy

Python and C++:
- Free, multiplatform
- Object-oriented
- Mostly stable

Python:
- Flexible
- Dynamic (“duck typing”)
- Many available libraries

SWIG

C++:
- Many standard tools
- Fast executables
- Even more libraries

Architecture

Size:

```
% date
% Thur Aug 24 9:41:40 EDT 2006
% cd SRC
% find . -name '*.py' | xargs wc | tail -1
  99609  338645  3812236 total
%
% find . -name '*.C' -o -name '*.h' | xargs wc | tail -1
  61359  218636 1875088 total
%
% find . -name '*.swg' | xargs wc | tail -1
   8839  27754  239741 total
%_
```

- Raw line-count includes duplicate disclaimers, import lines, comments, etc.
- Source tree traversal omits testing code, worked examples, prototypes, etc.

Architecture

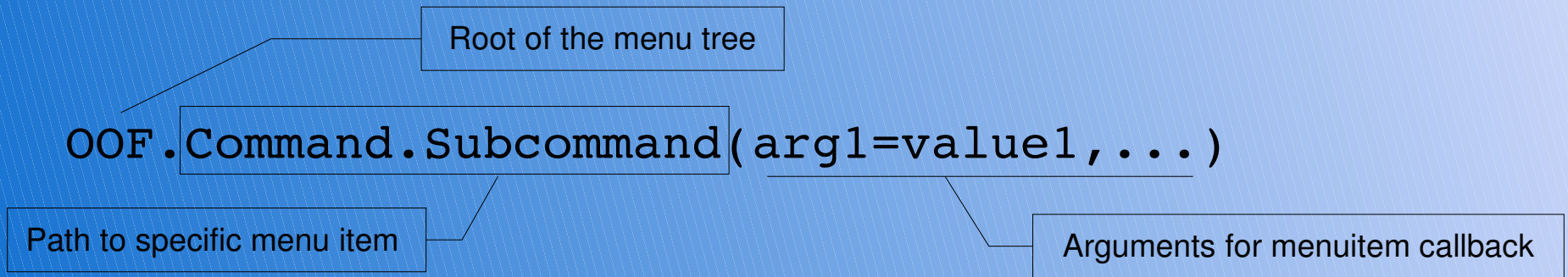
Dependent packages/libraries:

- SparseLib++ (integrated into source)
- Distutils (included with Python)
- SWIG (build-time only)
- PyGTK
  - GTK2
- GnomeCanvas
- ImageMagick
- Blas/lapack
- Docbook
- MPI/PETSc

} Not required to run current release

Architecture

All consequential operations are expressed in terms of commands from a hierarchical menu



This command locates the menuitem in the tree, and invokes its callback with the item itself as the first argument, appending the specified keyword arguments

Architecture

The menuitem tree is extensible at runtime. Some objects, when created, create their own submenus

GUI operations construct and call menu items

- GUI operations generate a script
- Script is replayable in non-GUI mode

Future: Parallel back-ends will be operated by menu commands

Architecture

Parameters:

- Sophisticated type-aware containers for arguments to menu items, some object constructors
- Automatic generation of widgets to fill in values
- Capable of convertibility
  - Object may have different representations
    - e.g. RGB vs. HSV vs. Grayscale for colors

OOF supports a high level of extensibility through automatic generation of widgets for arguments to menu commands

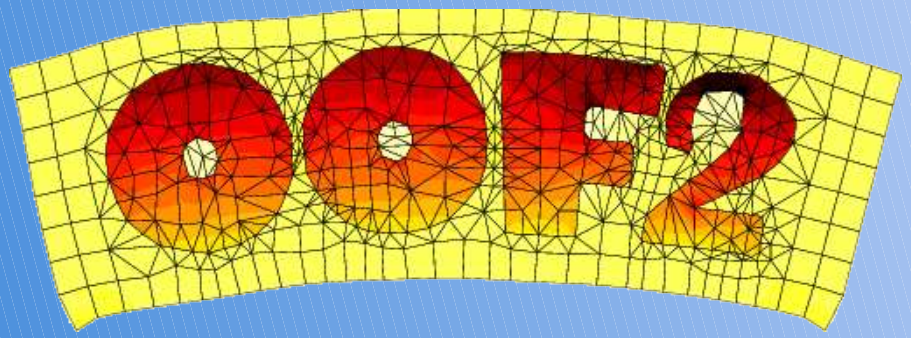
Architecture

Python/C++ boundary

One advantage of the SWIG approach is that we can place the language boundary in different places according to flexibility or performance requirements

- GUI in Python (PyGTK linkage, user-time, flexibility)
  - Exception: GnomeCanvas (absence of wrappers, speed)
- Menu tree in Python (user-time, dynamic option assignments)
- Skeleton object in Python (flexibility, complex undo/redo)
  - Possible peformance problem, may be changed
- FE physics represenation in C++ (maturity, speed)
- FE matrix construction in C++ (SparseLib++ linkage, speed)

OOF2 Summary



- Modular open-source tool for materials modeling
  - Users explore structure-property relationships
- Friendly to complex microstructural geometries
- Constitutive rules expressed in materials-science terms
- User extensibilty of constitutive rules
- User-friendly GUI, command-line, or remote operating modes
- Python scriptability – loops and conditionals
  - “virtual experiments”