Argonne NATIONAL LABORATORY

Data Models & their Influence On…

Tim Tautges
Argonne National Lab

CSCADS Workshop
July 31, 2012

U.S. DEPARTMENT OF ENERGY

What is a Data Model?

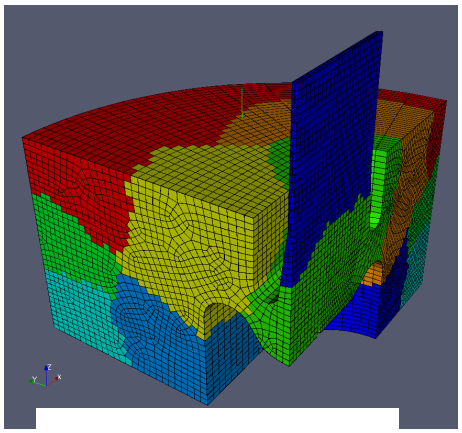
- What is a data model?
  - In language, a data model is analogous to the words used to tell your story
  - In code, the data structures used in composing algorithms
  - In a library, the data types used to communicate with the library
- Here, I'm concerned with libraries
- Why is a library's data model important?
  - Strongly affects usability of the library
  - Determines what can be expressed through the library's API
- Characteristics of a good data model
  - Balanced between concreteness and abstractness
    - Too concrete: code gets too verbose
    - Too abstract: code difficult to understand
  - Abstractions cover current *& future* needs
- In library design, once you've determined scope and data model, API should fall out naturally

Examples of Data Models (& their problems)

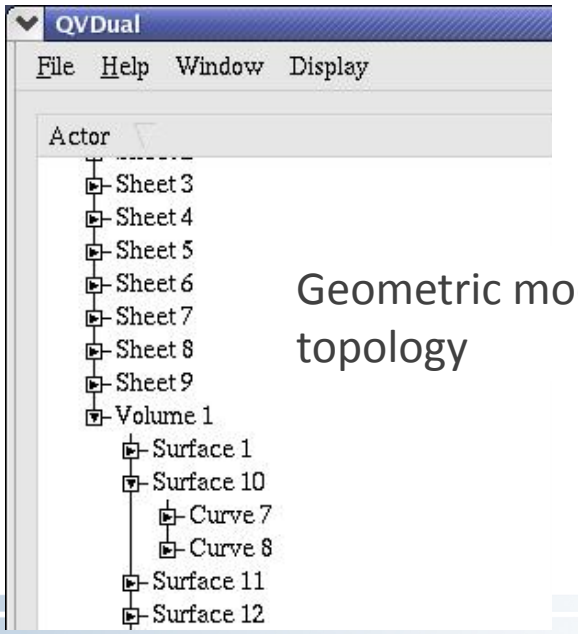
- ExodusII
  - Format for storing FEA mesh, analysis results
  - Node, Element, Element Block, Sideset, Nodeset, variable, timestep
  - *Element block represents both material and fundamental element*
  - *No mechanism for defining other groupings of elements, e.g. proc decomposition, AMR tree, etc.*
- CGNS
  - Fundamental element types TRI3, TRI6, HEX8, HEX20, etc.
  - Basic operation: get all hex elements
    - Get hex8, get hex20, get hex27

ITAPS Data Model

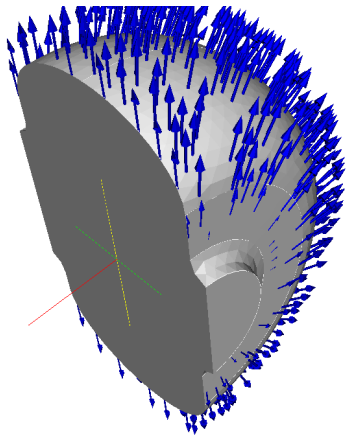
- Entities
  - Vertex, Edge, Tri, Quad, (Pentagon?), (Hexagon?), Polygon, Tet, Pyramid, Prism, Knife, Hex, Polyhedron
- Sets (collections of entities & sets, parent/child links)
  - BC groups, materials, proc partitions, kdtree nodes, …
- Interface (OOP, owns data)
- Tags (annotation of data on other 3)
  - Fine-grained (entities): vertex-based temperature, element-based heat generation rate
  - Coarse-grained (sets, interface): BC type, proc rank, provenance



Parallel Partition



Geometric model topology



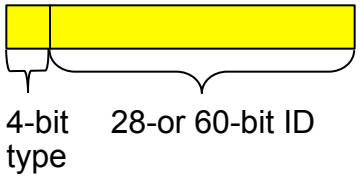
Vertex-based displacements

Mesh-Oriented datABase (MOAB)

- Library for representing, manipulating structured, unstructured mesh models
- Supported mesh types:
  - FE zoo (vertices, edges, tri, quad, tet, pyramid, wedge, knife, hex)
  - Polygons/polyhedra
  - Structured mesh
- Optimized for memory usage first, speed second
- Implemented in C++, but uses array-based storage model
  - Avoids C++ object-based allocation/deallocation
  - Allows access in contiguous arrays of data
- Mostly an ITAPS-like data model
  - Entity, set, tag, interface
- Mesh I/O from/to various formats
  - HDF5 (custom), vtk, CCMIO (Star CD/CCM+), Abaqus, CGM, Exodus
- Main parts:
  - Core representation
  - Tool classes (skinner, kdtree, OBBtree, ParallelComm, …)
  - Tools (mbsize, mbconvert, mbzoltan, mbcoupler, …)

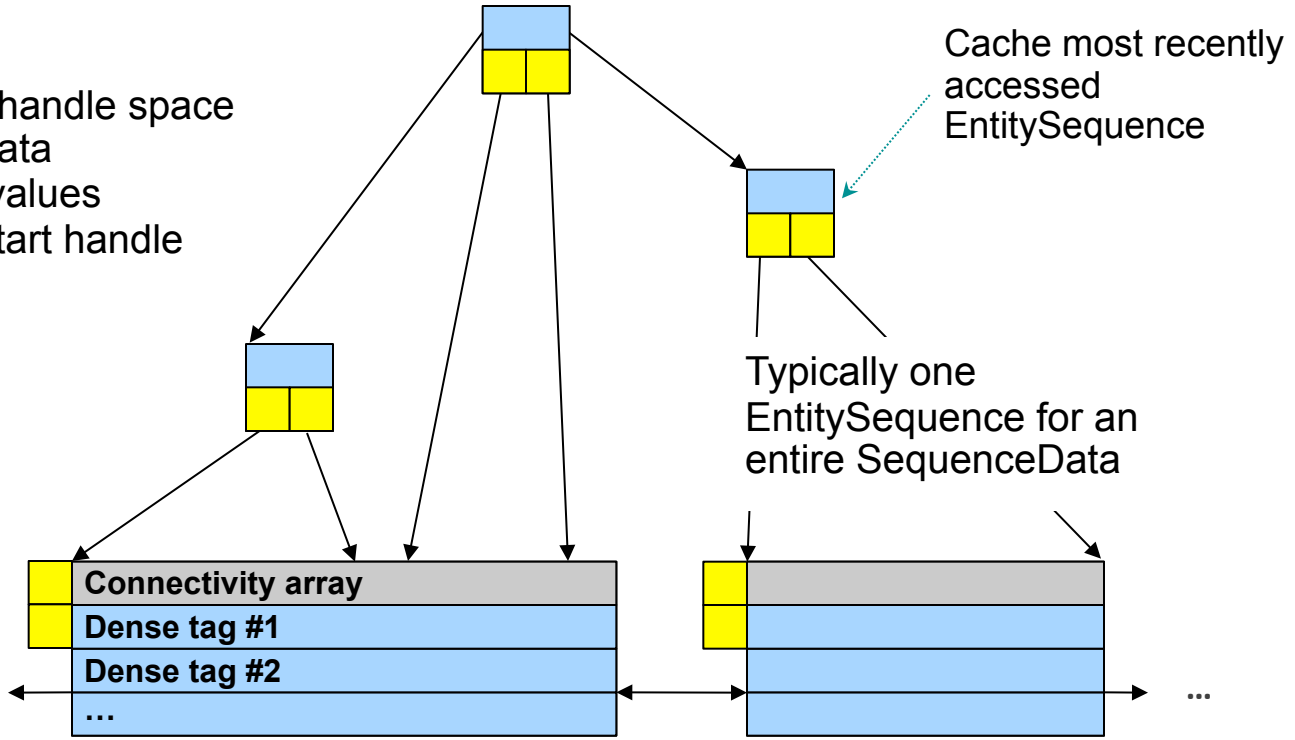
MOAB Entity Storage

Entity Handle:
- Unsigned long type
- Bitmask
- Sorts by dimension, type



Range:
- Container of handles
- Constant-size if contiguous handles

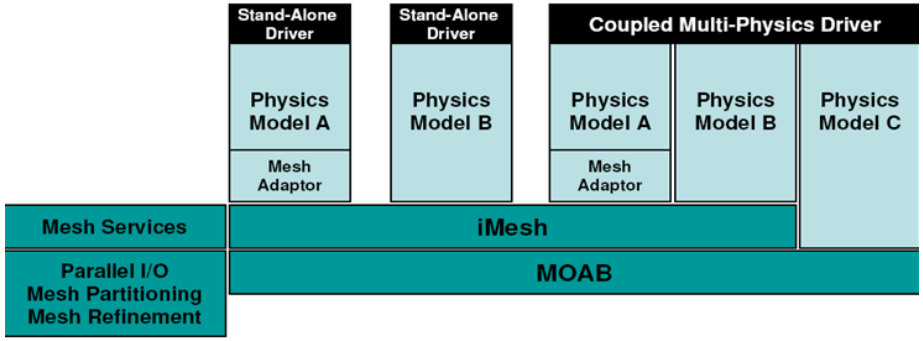
EntitySequences:
- Represent *used* portions of handle space
- Have pointer to SequenceData
- Have start and end handle values
- Arranged in binary tree by start handle



SequenceData:
- Represent *allocated* portions of handle space
- Have start and end handle
- Coordinates or Connectivity
- Dense Tag Data

Influence on Computation…

- Mesh acts as a vehicle for much other simulation data
- Pursuing various efforts to use MOAB as a simulation data backplane, e.g.
  - NEAMS/CESAR



- Requires:
  - Reading/initializing mesh from MOAB
  - Pushing simulation results down into MOAB
- Under certain conditions, MOAB can share field data directly with application, as contiguous-memory arrays
  - *But,* requires app and MOAB to use the same local ordering
  - In practice, apps come with their own expectations about ordering
  - Will require local reordering in MOAB (will also be useful for on-node shared memory)

Influence on I/O…

- Current HLL for I/O interact in terms of 1D or multi-D arrays
  - Translation from various grid-based data structures can be non-trivial amount of code, even for common ones (“7 dwarves”), e.g. unstructured, structured AMR, particle, etc.
- Need HLL’s that communicate at a higher level of abstraction
- Damsel project: present a HLL for I/O in terms of grid and grid-based data
  - Reduce the “impedance mismatch” between apps and I/O library
  - Minimize data copies between app & storage
  - Enable other operations on data in-flight, e.g. compression, query

Influence on Data Analysis…

- Assertion: many important pieces of an integrated data analysis capability are either available as components of original simulation codes, or are being made available as components; i.e. analysis and simulation are converging in the tools they use
  - Data I/O, representation, access
  - Numerical operators (max/min, gradient)
  - Others (MS complex, streamlines, etc.)
- This is a Good Thing in terms of both code reuse and accuracy
- Using well-thought-out data model makes that easier, since components are more flexible & have more headroom

Influence on Visualization…

- If I can viz the data model, I can viz the data
- For example:

Currently:

Special readers/datastructures for:
Geometric model, Boundary conditions, Processor partitions, …

+

Special handling for:
Picking, drawing, filtering, GUI form interactions, …

Moving toward:

Common set/tag conventions for:
Geometric model, Boundary conditions, Processor partitions, …

+

Common handling for:
Picking, drawing, filtering, GUI form interactions, …

- Moving towards that as part of mesh generation SBIR with Kitware; should have VTK-based entity sets by end of Aug
- Then it becomes a question of how best to abstract any given data
  - E.g. spectral element mesh/data

Going Forward…

- Even if we accomplish this component-based vision, some difficult questions remain
  - What to do about overlaps between existing components/libraries
    - E.g. in CESAR, between MOAB, DIY, GLEAN
  - Are there any critical component prototypes missing, such that everyone will have to wait for that before they have a good integrated solution?
    - Materials? Discretization? Fields?
  - Can we develop benchmarks for pieces that can be composed into a full benchmark too?